**Instituto Tecnológico de Aeronáutica - ITA**
**Inteligência Artificial para Robótica Móvel - CT-213**
**Aluno: Rafael Mello Celente**

# Relatório do Laboratório 9 - Detecção de Objetos

## 1. Breve Explicação em Alto Nível da Implementação

Para a implementação do modelo foi utilizado o framework do *Keras*, montando uma arquitetura de rede pré-estabelecida que possui 9 camadas, além de uma camada residual. A arquitetura implementada é mostrada na Figura 1.

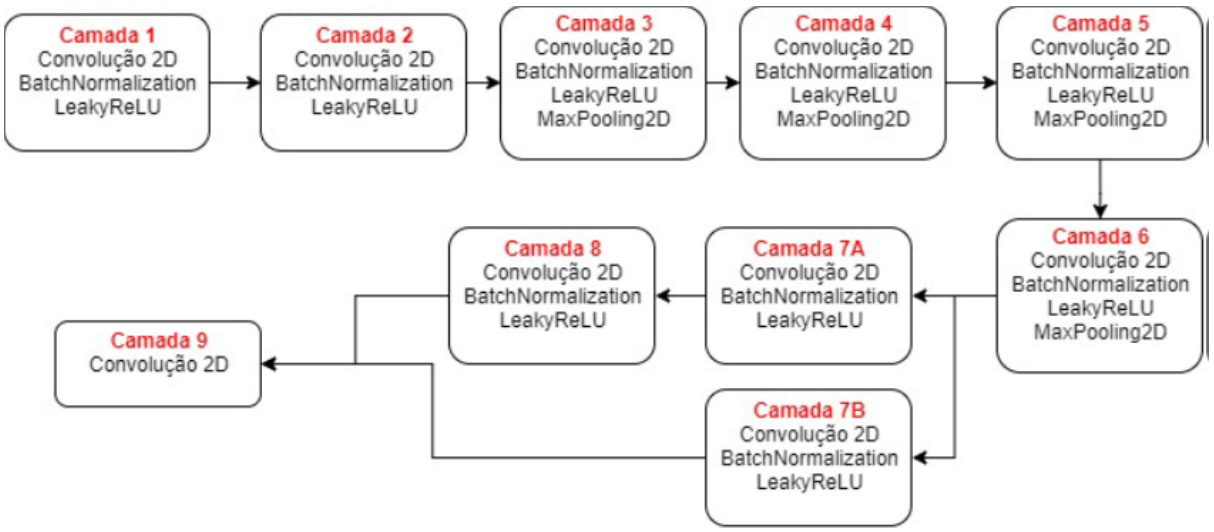
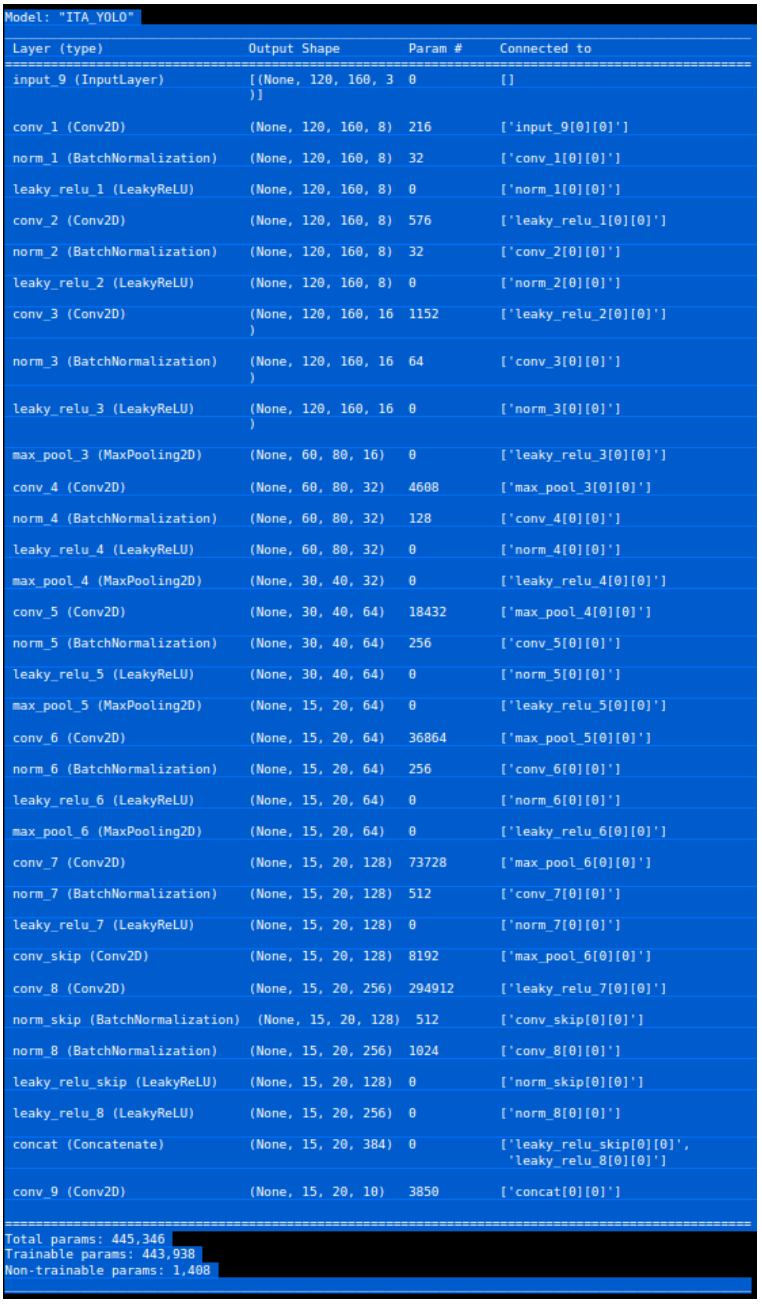



**Figura 1.** Arquitetura de rede utilizada.

Apesar da implementação da arquitetura, ela não foi de fato treinada para economizar tempo, uma vez que seu treinamento demoraria cerca de 8h mesmo em um GPU de alto desempenho. Portanto, o modelo já treinado foi utilizado, aplicando apenas a parte de inferência no laboratório.

Essa parte de inferência, por sua vez, foi implementada a partir do desenvolvimento de 3 funções: *detect, preprocess_image* e *process_yolo_output.* A função *preprocess_image* tinha por objetivo redimensionar a imagem, transformá-la em um *array* e finalmente, normalizar os canais de cor para melhorar o desempenho da rede. Já a função *process_yolo_output* tinha por objetivo retornar, a partir do vetor de features de cada uma das células, a probabilidade, localização e tamanho das *bounding boxes* para a bola e para as traves, retirando da matriz de todas as células a que tinha maior probabilidade de ser a bola, e as duas maiores probabilidades de serem as traves. Finalmente, a função *detect* tinha por objetivo executar a função *preprocess_image*, seguido de predizer a imagem através do modelo, e por último, executar a função *process_yolo_output.*

### 1.1. Sumário do Modelo



```
Model: "ITA_YOLO"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_9 (InputLayer)        [(None, 120, 160, 3       0         []
                             )]

 conv_1 (Conv2D)             (None, 120, 160, 8)       216       ['input_9[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
_________________________________________________________________________________________________
```

## 2. Figuras Comprovando Funcionamento do Código

### 2.1. Detecção de Objetos com YOLO

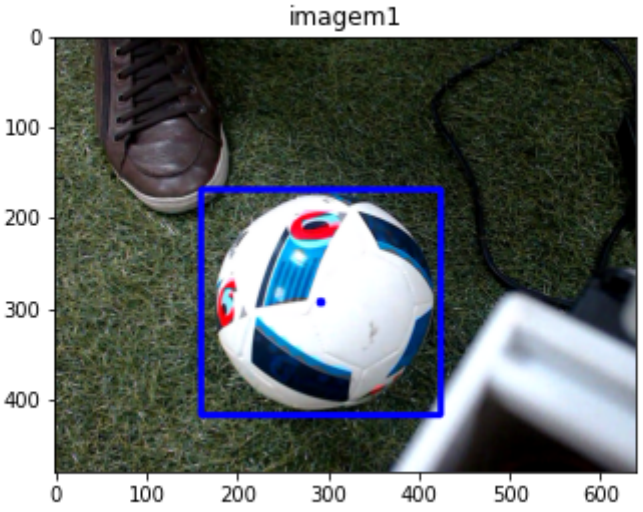


imagem1
0
100
200
300
400
0
100
200
300
400
500
600

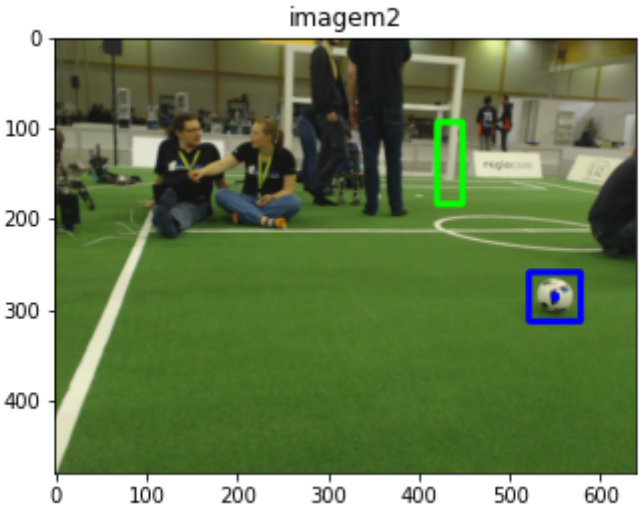


imagem2
0
100
200
300
400
0
100
200
300
400
500
600

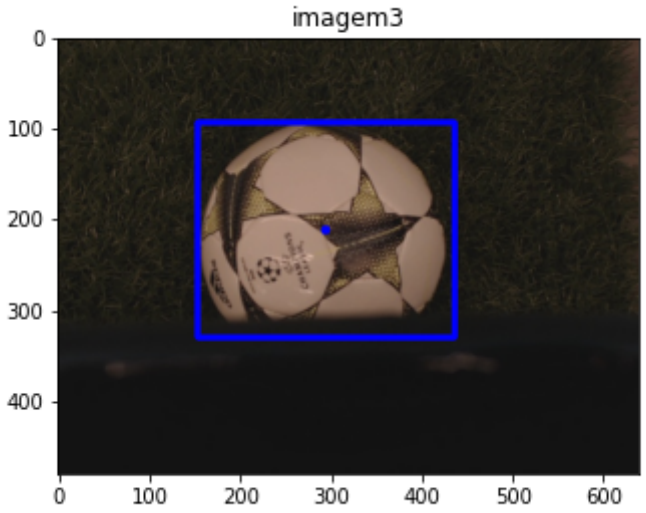


imagem3
0
100
200
300
400
0
100
200
300
400
500
600

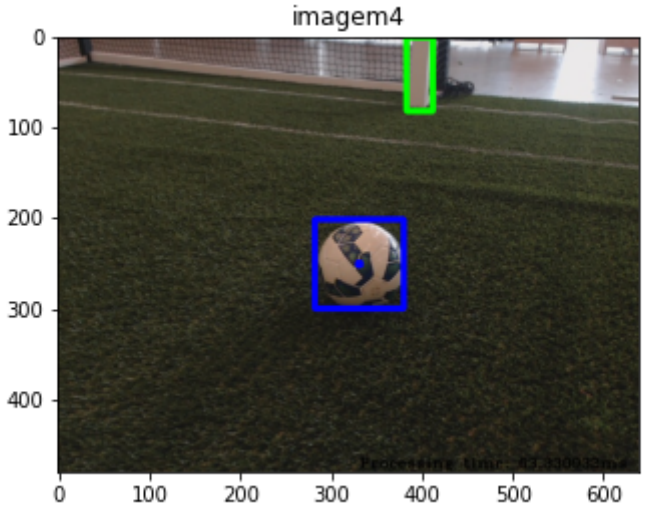


imagem4
0
100
200
300
400
0
100
200
300
400
500
600

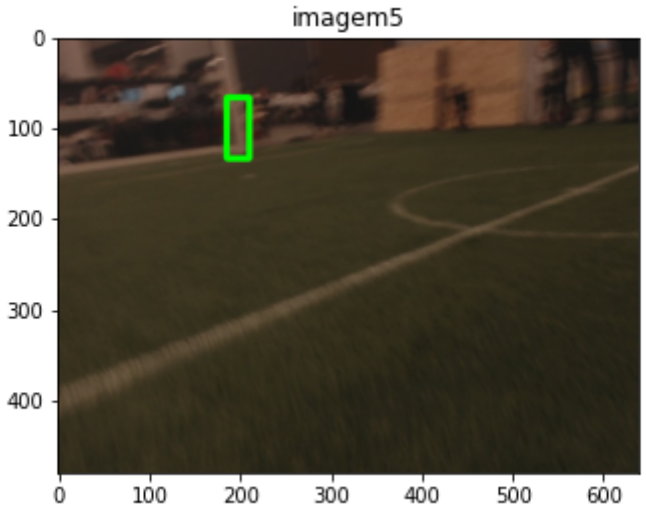


imagem5
0
100
200
300
400
0
100
200
300
400
500
600

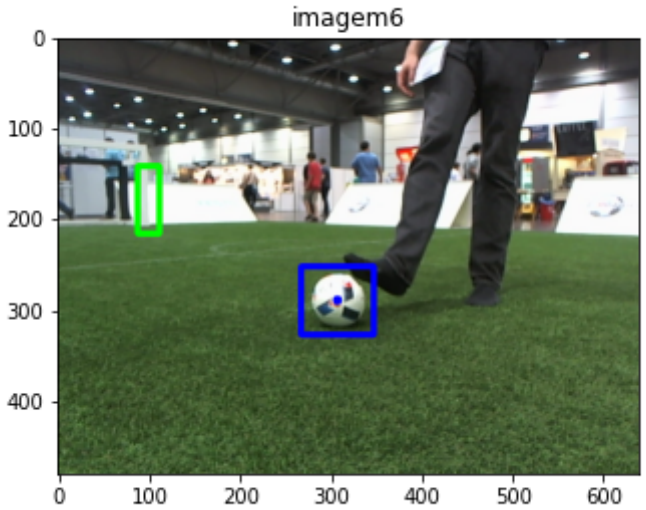


imagem6
0
100
200
300
400
0
100
200
300
400
500
600

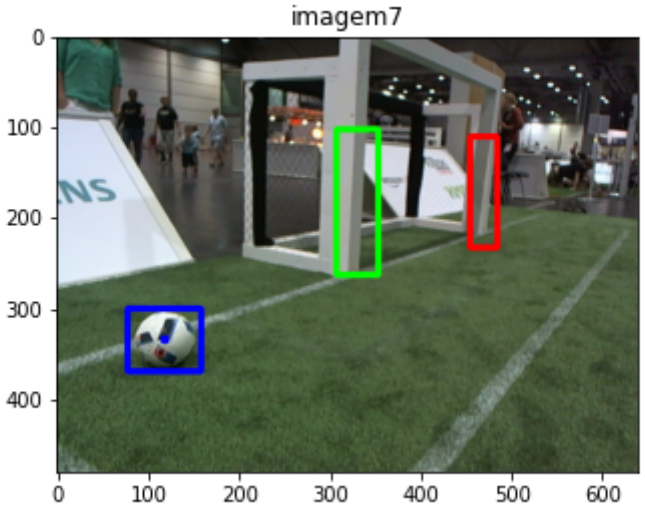


imagem7
0
100
200
300
400
0
100
200
300
400
500
600

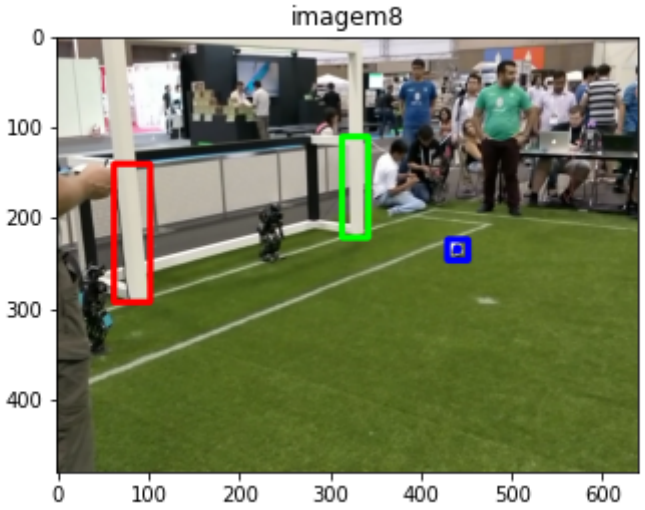
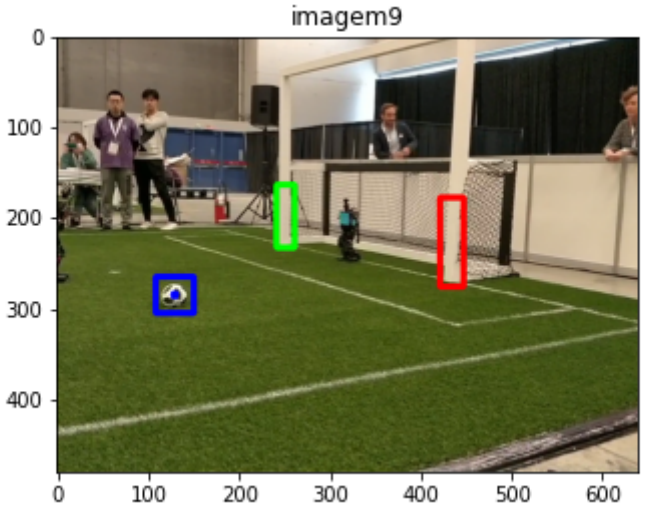
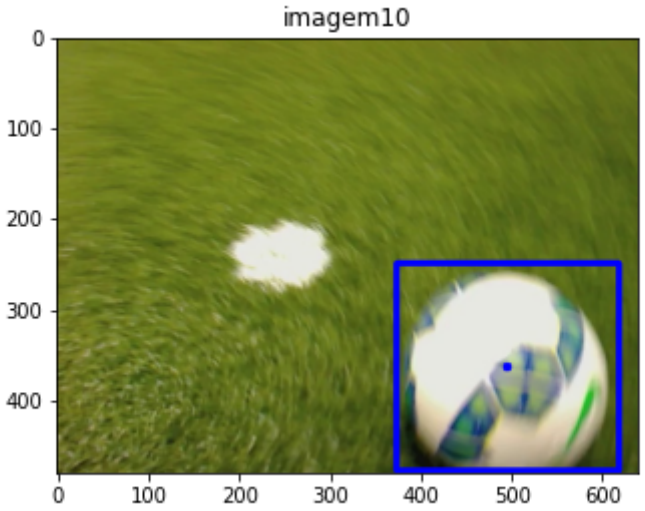


imagem8
0
100
200
300
400
0
100
200
300
400
500
600




## 3. Discussão

Pode-se perceber que o método utilizado baseado no YOLO é extremamente efetivo em detectar os itens de interesse com precisão, definindo suas *bounding boxes* de forma efetiva mesmo em distâncias e iluminações diferentes. A eficácia do método também é demonstrada não somente no que o modelo consegue detectar, mas naquilo que ele consegue não detectar também, uma vez que muitas partes das imagens testadas tinham linhas e círculos brancos que poderiam facilmente serem confundidos por bolas ou traves pelo modelo.