

# **Inteligência Artificial para Robótica Móvel**

**Visão usando RNC**

**Professor: Marcos Maximo**

# Roteiro

- Motivação;
- Classificação e Localização;
- Detecção de Objetos;
- *Sliding Windows* com CNN;
- YOLO;
- Estudo de Caso: Visão do Humanoide.

# Motivação

# Classificação x Detecção

## Classification



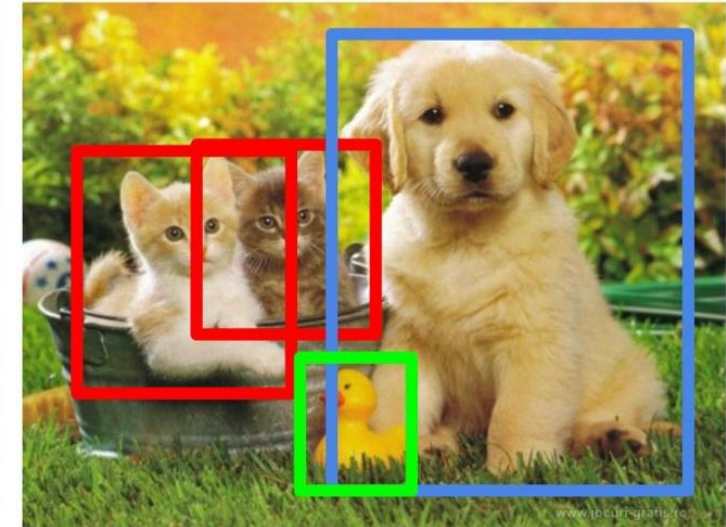
CAT

## Classification + Localization



CAT

## Object Detection

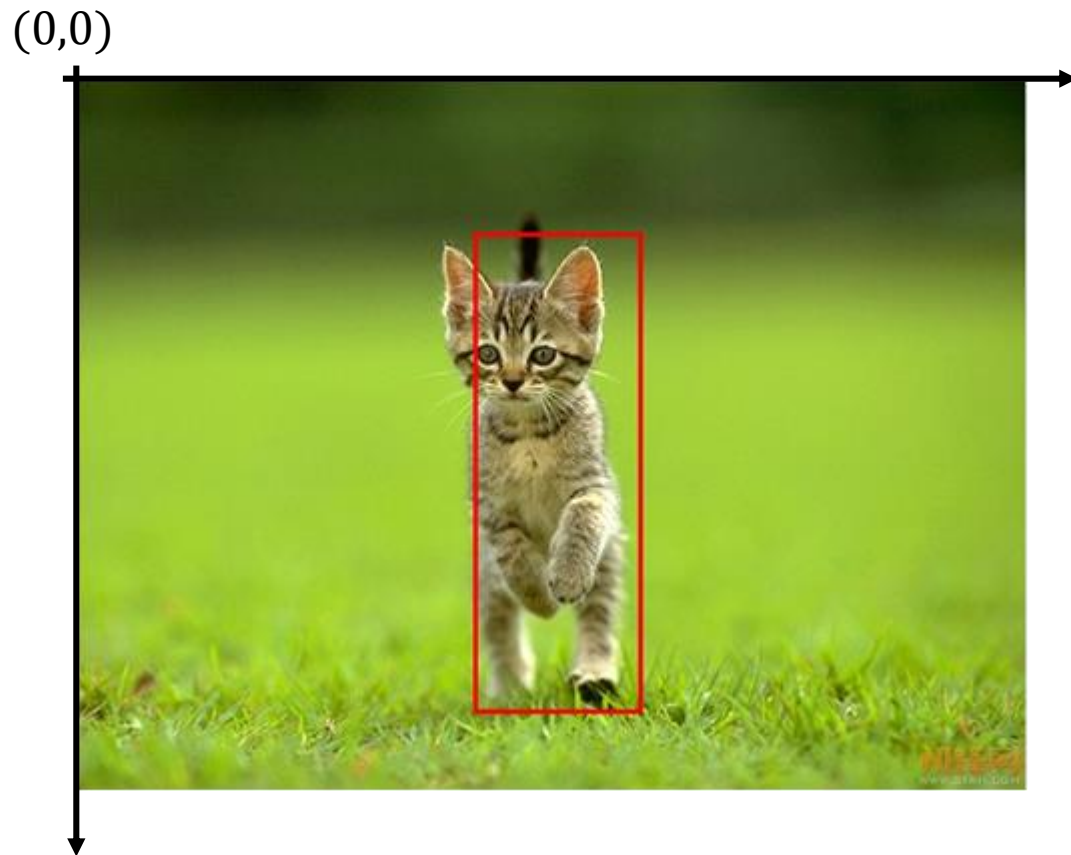


CAT, DOG, DUCK

Fonte: <https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>

# Classificação e Localização

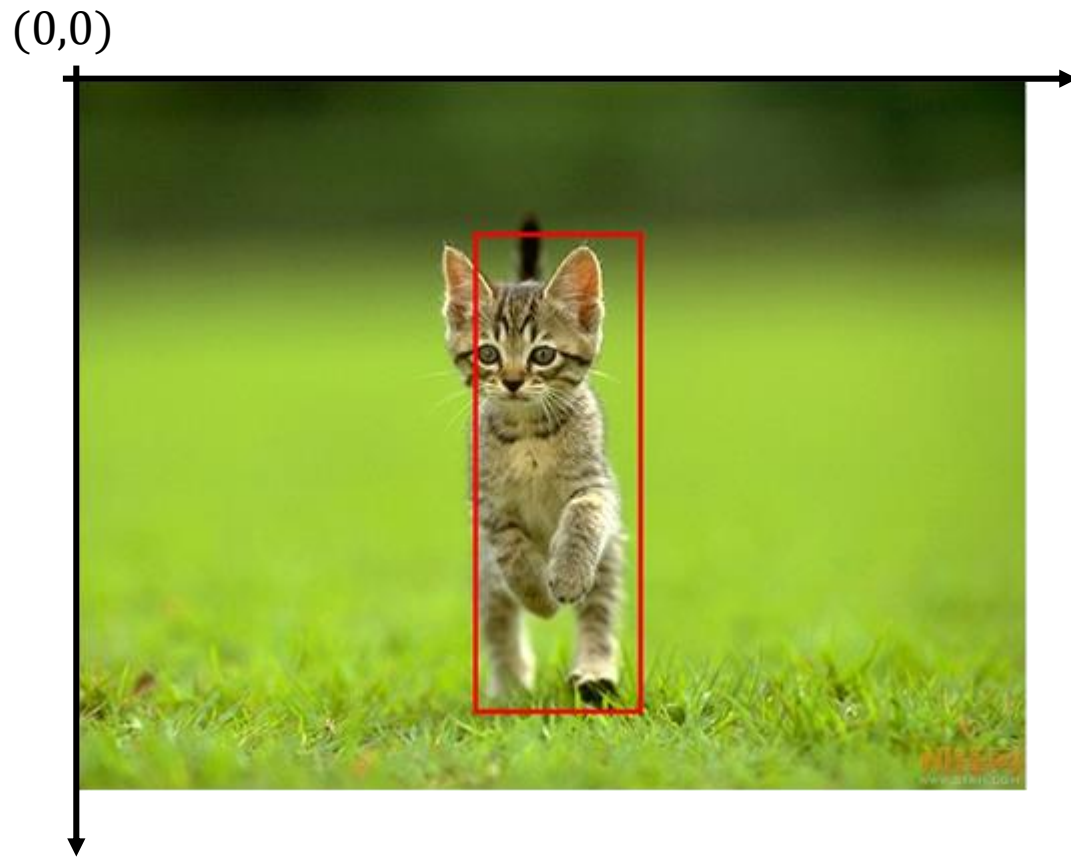
# Classificação com Localização



- $p_c$ : presença do objeto.
- $(b_x, b_y)$ : centro do *bounding box*.
- $(b_w, b_h)$ : dimensões do *bounding box*.
- Feature vector:

$$\mathbf{y} = [p_c, b_x, b_y, b_w, b_h]^T$$

# Classificação com Localização



- Com várias classes:

$$\mathbf{y} = [p_c, b_x, b_y, b_w, b_h, c_1, c_2, c_3]^T$$

- *Loss function:*

- $p_c = 1$ :

$$\mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \sum_{j=1}^{n_y} \left( \hat{y}_j^{(i)} - y_j^{(i)} \right)^2$$

- $p_c = 0$ :

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = (\hat{p}_c - p_c)^2$$
$$\mathbf{y} = [0, ?, ?, ?, ?, ?, ?, ?]^T$$

# Landmark Detection



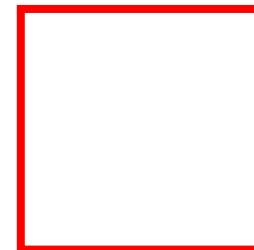
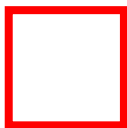
- Pode-se incluir várias *features* (pontos) no vetor de *features*, além do *bounding box*:

$$[l_{1x}, l_{1y}, l_{2x}, l_{2y}, \dots, l_{nx}, l_{ny}]^T$$



# Detecção de Objetos

# *Sliding Window Detection*



# *Sliding Windows Detection*

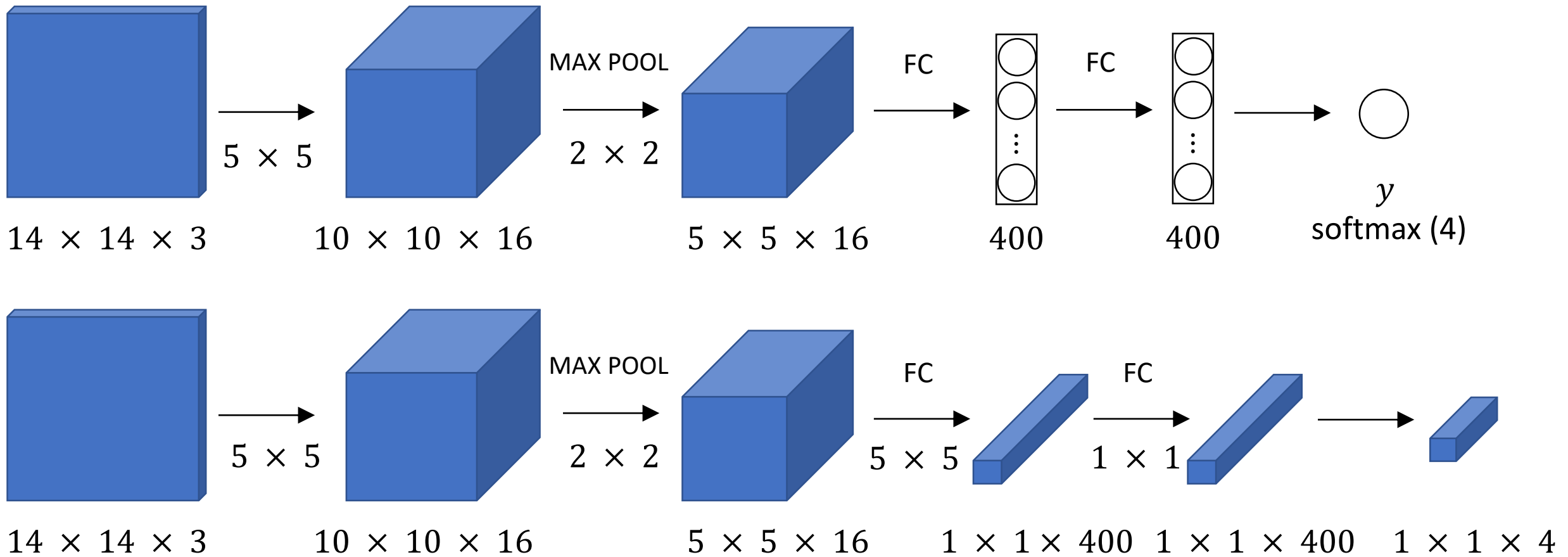
- Jeito clássico de se fazer antes de *Deep Learning*.
- Custo computacional muito elevado.
- Precisão da localização não é muito boa, pois desloca-se a janela em passos discretos.

# *Region of Interest (ROI)*

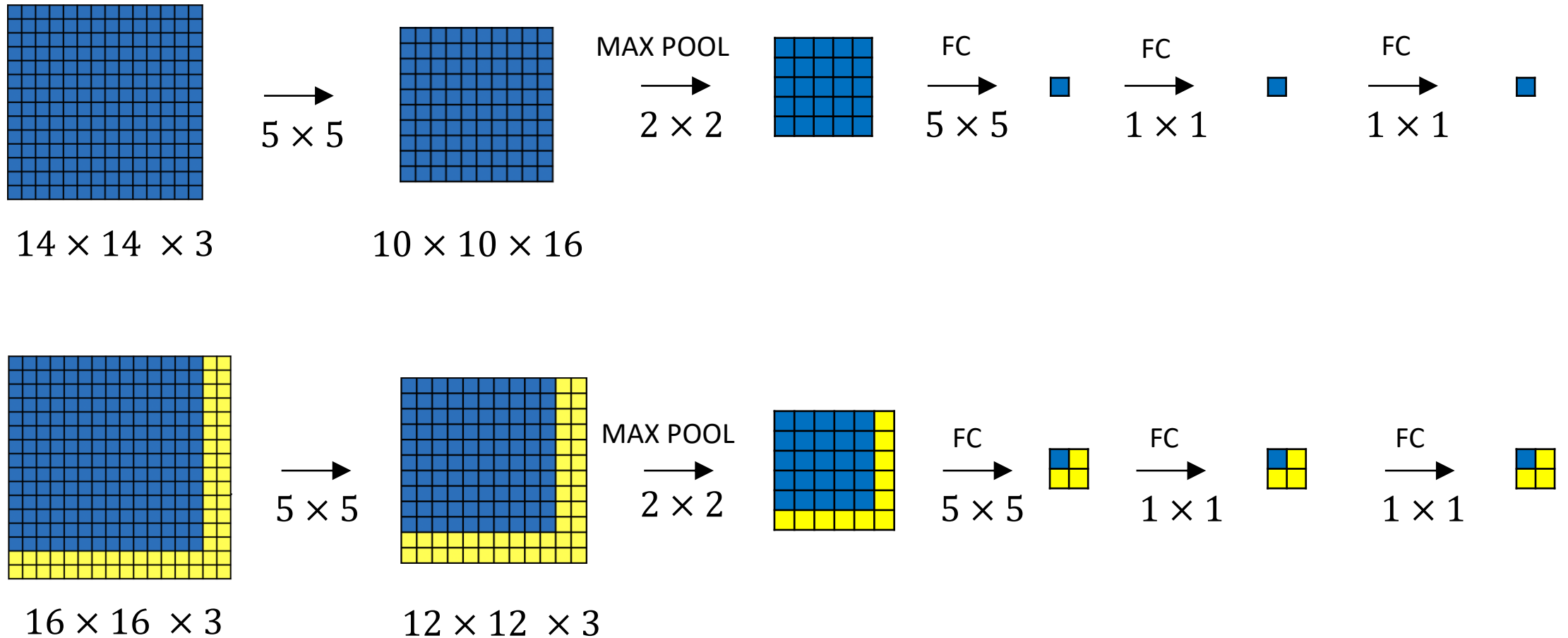
- Outra maneira de posicionar a janela é por região de interesse (ROI).
- Primeiro, executa-se um algoritmo mais simples para detectar candidatos a serem o objeto.
- Em geral, usa-se algoritmos de visão clássica.
- Um classificador baseado em aprendizado precisa apenas “filtrar” os candidatos ao invés de buscar na imagem inteira.

# *Sliding Windows* com CNN

# Implementação de FC com CONV



# Sliding Windows com CNN



YOLO

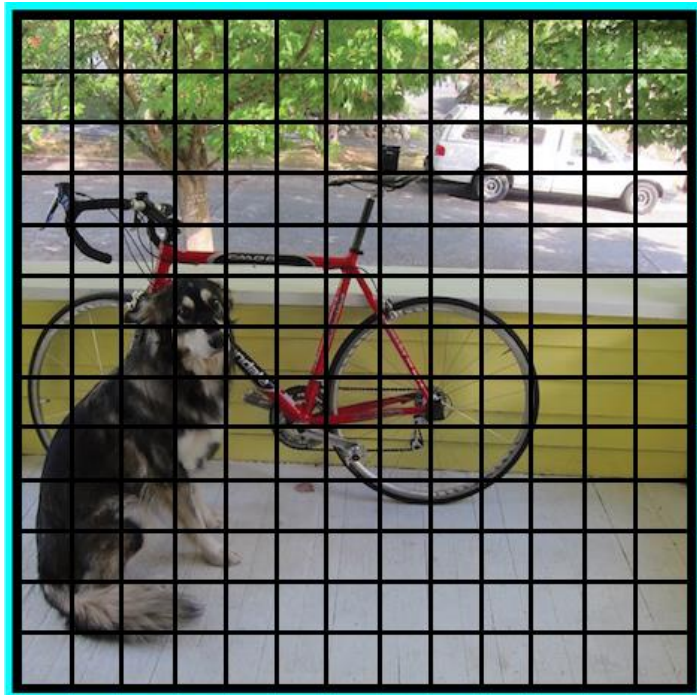


# YOLO

- Inglês: *You Only Look Once* (YOLO).
- Como o nome indica, tenta resolver o problema de várias inferências em *sliding windows*.
- Algoritmo de detecção de objetos baseado em CNN.
- CNN baseada na GoogLeNet.
- Atualmente, já está na versão 3.

# YOLO

- Ideia: dividir a imagem em um grid  $S \times S$  (e.g. 19x19).
- Associar a cada célula um vetor de *features*.



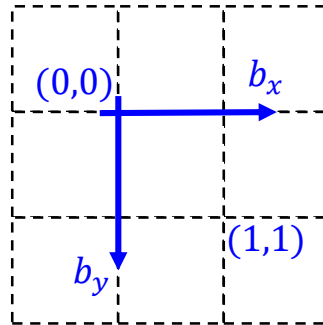
$$\mathbf{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$$

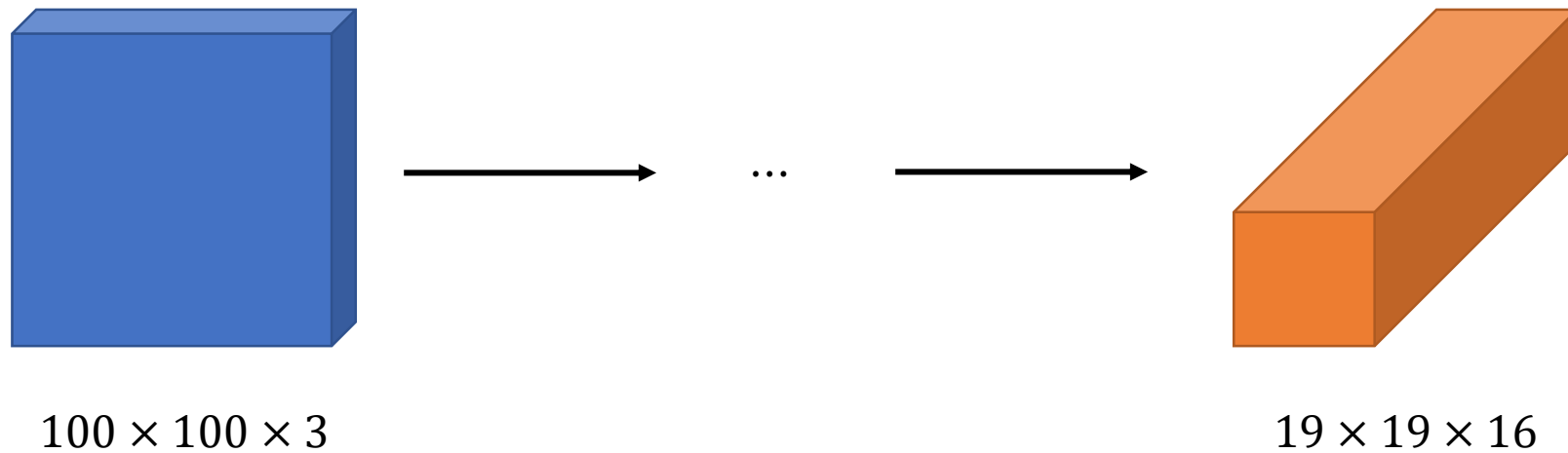
# YOLO

- $(b_x, b_y)$  em relação à célula (e normalizado de 0 a 1).
- $(b_w, b_h)$  em relação à imagem.
- Precisa transformar no treinamento e na inferência.
- Versão 2 usa um método mais complicado.

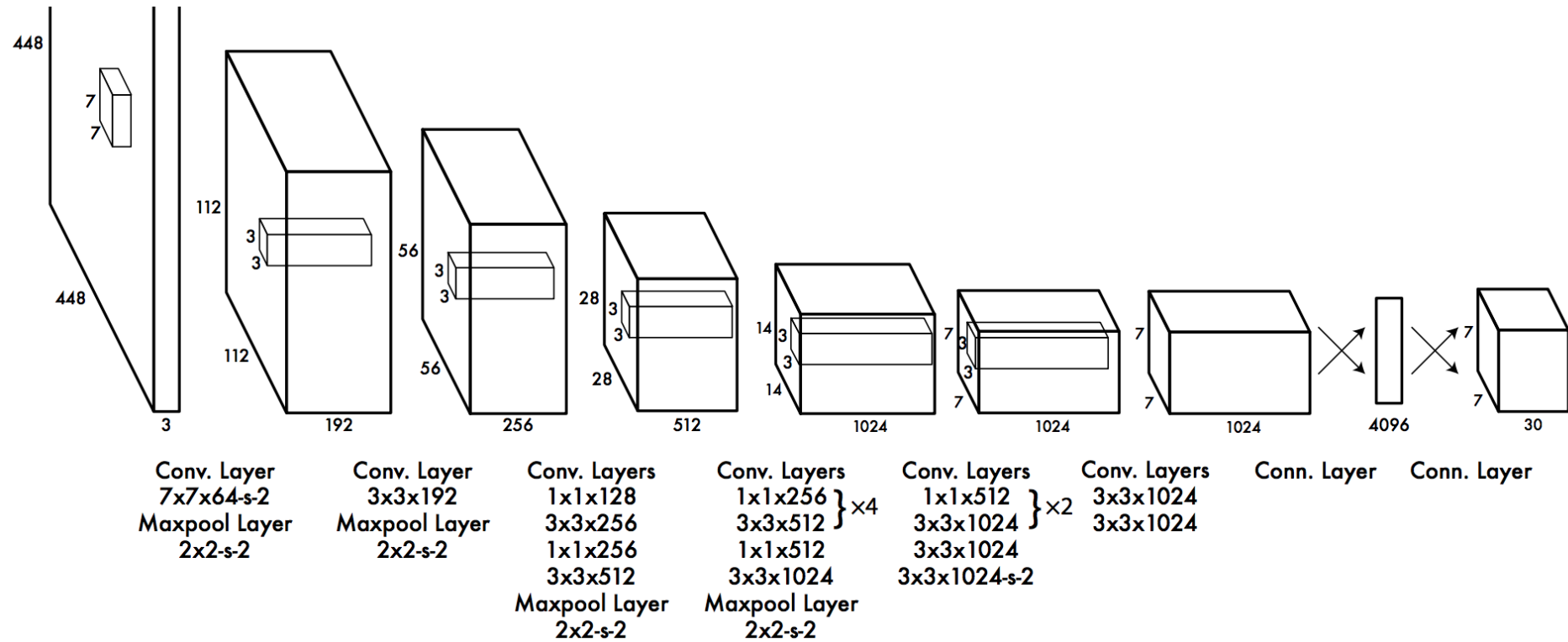


# YOLO

- Como há um vetor de features para cada célula, a saída é  $S \times S \times n_y$ .



# CNN do YOLO



# CNN do YOLO

- YOLO: 24 camadas convolucionais e 2 totalmente conectadas.
- FastYOLO: 9 camadas convolucionais e 2 totalmente conectadas.
- Camadas da FastYOLO também tem menos filtros.


# CNN do YOLO

- Última camada usa função de ativação linear.
- Todas as outras usam Leaky ReLU:

$$\phi(x) = \begin{cases} x, & x > 0 \\ 0, & 1x, x \leq 0 \end{cases}$$

# *Intersection over Union (IoU)*

- Métrica para avaliar *bounding box*.
- Alinhamento perfeito:  $IoU = 1$ . Pior alinhamento:  $IoU = 0$ .
- Em geral, se  $IoU \geq 0,5$  (arbitrário), então localização “correta”.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Fonte: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>



# Confidence

- No lugar de apenas a probabilidade do objeto estar presente ( $p_c$ ), YOLO sugere usar *confidence*:

$$C = \text{Pr}(\text{Object}) * \text{IoU}(\text{pred}, \text{truth})$$

- YOLO também mantém apenas um vetor em cada célula para indicar as classes presentes:

$$\mathbf{p}_i = \begin{bmatrix} p_i(c = 1) \\ p_i(c = 2) \\ p_i(c = 3) \end{bmatrix}$$

- Em tempo de execução, calcula-se *confidence* por classe:

$$C_{i,j}(c) = C_{i,j} * p_i(c)$$

# *Confidence*

- Vetor de *features* fica:

$$\mathbf{y} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{p} \end{bmatrix}$$

- Bounding box definido por:

$$\mathbf{b} = [C, x, y, w, h]^T$$

# YOLO

- *Loss function:*

$$\begin{aligned} \mathcal{L} &= \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{obj} [(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2] \\ &+ \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{obj} \left[ (\sqrt{\hat{w}_i} - \sqrt{w_i})^2 + \left( \sqrt{\hat{h}_i} - \sqrt{h_i} \right)^2 \right] \\ &+ \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{obj} (\hat{C}_i - C_i)^2 + \lambda_{noobj} \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{noobj} (\hat{C}_i - C_i)^2 \\ &+ \sum_{i=1}^{S^2} \varepsilon_i^{obj} \sum_{c=1}^C (\hat{p}_i(c) - p_i(c))^2 \end{aligned}$$

# YOLO

- Se objeto está presente:

$\mathcal{L}$

$$\begin{aligned} &= \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{obj} [(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2] \\ &+ \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{obj} \left[ \left( \sqrt{\hat{w}_i} - \sqrt{w_i} \right)^2 + \left( \sqrt{\hat{h}_i} - \sqrt{h_i} \right)^2 \right] \\ &+ \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_i^{obj} (\hat{C}_i - C_i)^2 + \sum_{i=1}^{S^2} \varepsilon_{i,j}^{obj} \sum_{c=1}^C (\hat{p}_i(c) - p_i(c))^2 \end{aligned}$$

# YOLO

- Se objeto **não** está presente:

$$\mathcal{L} = \lambda_{noobj} \sum_{i=1}^{S^2} \sum_{j=1}^B \varepsilon_{i,j}^{noobj} (\hat{C}_i - C_i)^2$$

# YOLO

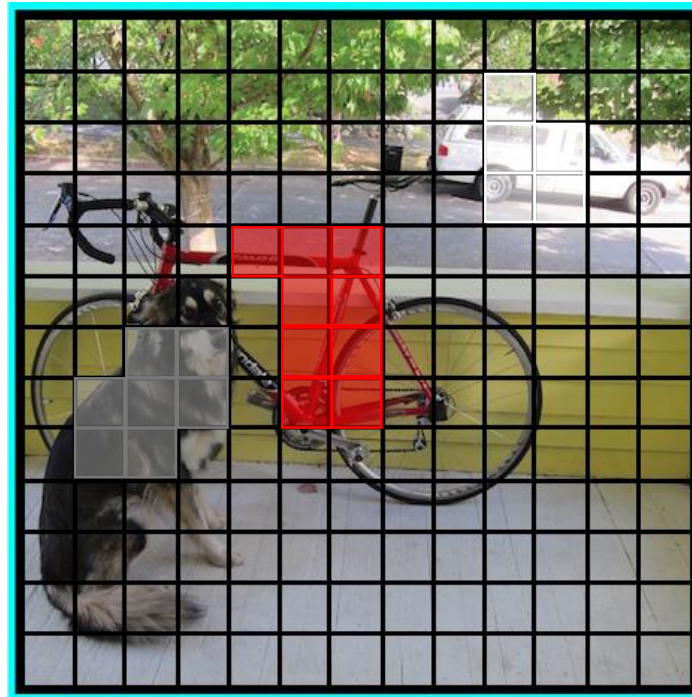
- A *loss function* é quadrática, mesmo para os termos de “classificação”.
- Raiz quadrada em  $w$  e  $h$  para penalizar menos erros grandes em *bounding boxes* grandes.
- $\lambda_{coord} = 5, \lambda_{noobj} = 0,5$ .
- $\varepsilon_{i,j}^{obj}$  se objeto presente na *bounding box*  $j$  da célula  $i$ .
- $\varepsilon_{i,j}^{noobj} = 1 - \varepsilon_{i,j}^{obj}$ .

# YOLO

- Durante o treinamento, para uma dada célula, como escolher a qual *bounding box* ( $\mathbf{b}_1$ ,  $\mathbf{b}_2$  etc.) do vetor de *features* associa-se um objeto?
- YOLO escolhe o *bounding box* com  $IoU(pred, label)$  máximo considerando os parâmetros atuais da rede.
- Isso faz com que *bounding boxes* fiquem especializados para certos tipos de objetos.

# *Non-max Supression*

- Objetos se estendem durante várias células.
- Múltiplas detecções do mesmo objeto.





# *Non-max Supression*

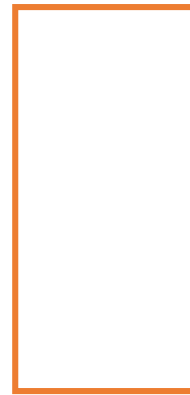
- Para cada classe:
  - Usar *threshold* inicial de *confidence* por classe para eliminar detecções pouco confiáveis.
  - Enquanto ainda houver detecção não tratada:
    - Pegar a detecção  $D_{max}$  com maior valor de *confidence* dentre as não tratadas.
    - Eliminar todas as detecções  $D_i$  com  $IoU(D_{max}, D_i) \geq 0,5$ .

# Anchor Boxes

- Definir de antemão qual o formato esperado de certo *bounding box*.
- Associar qual b.b. com base em  $IoU(anchor, label)$ .
- Ideia usada na YOLO v2.
- Definidos “na mão” ou com aprendizado não-supervisionado.



$b_1$



$b_2$

# YOLO v2

- Saída da rede é  $t_x, t_y, t_w, t_h$  e  $t_o$ . Então, calcula-se:

$$x = c_x + \sigma(t_x)$$

$$y = c_y + \sigma(t_y)$$

$$w = p_w e^{t_w}$$

$$h = p_h e^{t_h}$$

$$C = \sigma(t_o)$$

- $(c_x, c_y)$  é a origem da célula.
- $p_w$  e  $p_h$  são as dimensões do *anchor box*.
- $\sigma(.)$  é a função sigmóide. Seu uso garante que os valores em questão ficarão entre 0 e 1.

# YOLO v2

- CNN com 19 camadas convolucionais e 5 de maxpooling.

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

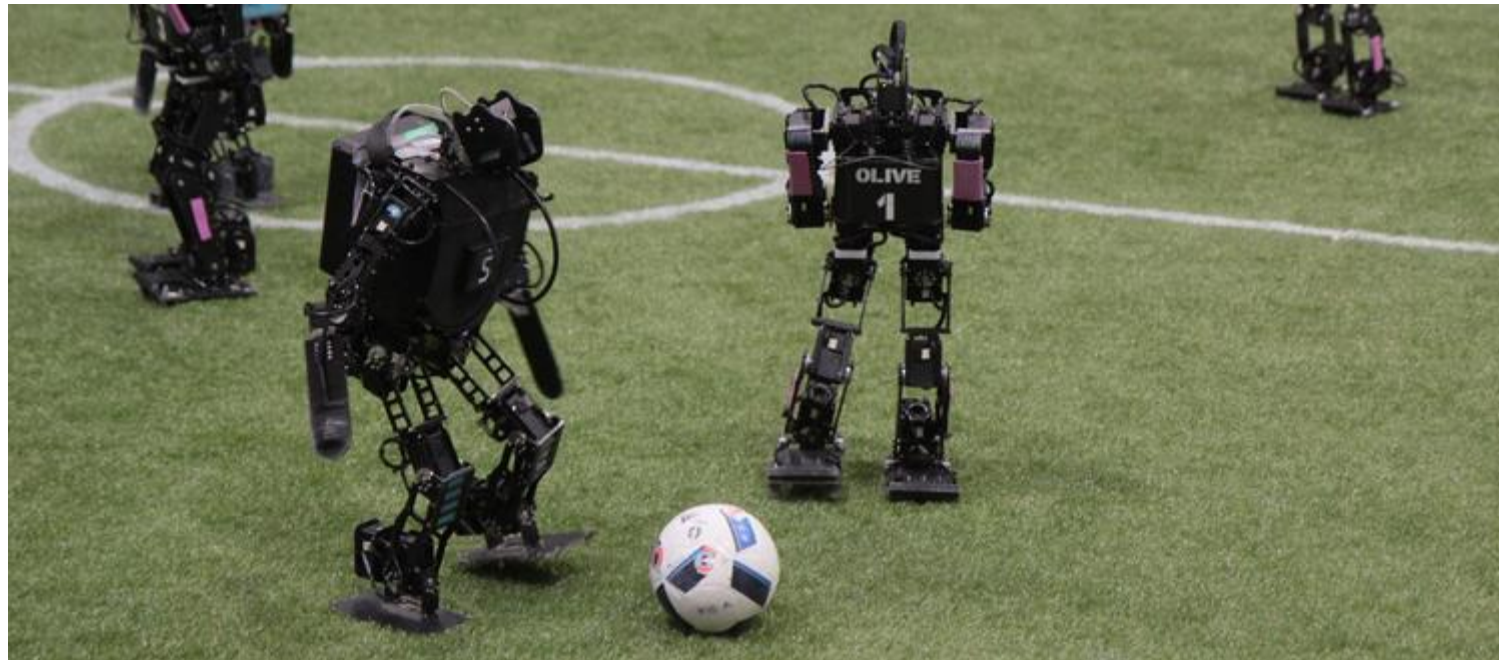
# YOLO v3

- Rede com 53 camadas.
- Uso de *Residual Networks* para permitir rede mais profunda.

# Estudo de Caso: Visão do Humanoide

# Visão do Humanoide

- Bola da RoboCup atualmente é branca.
- Confunde com outros objetos.



# Visão do Humanoide

- Iniciação Científica do Lucas Steuernagel (COMP-21).
- Inicialmente pensado para a bola.
- Extensão posterior para as traves (não conseguimos detectar bem com visão clássica).
- Baseado no YOLO.



# Visão do Humanoide

- Treinamento em Python com Tensorflow e Keras.
- Execução no robô em CPU (Intel Core i5) em C++ usando Tensorflow.
- Requisito de tempo no robô:  $< 100$  ms.
- Rede final executada em 60-80 ms no robô (CPU!).

# Visão do Humanoide

- Testou **13** redes diferentes até encontrar bom *trade-off* entre precisão e desempenho.
- Redução para 9 camadas.
- Redes usando 50% ou 25% da resolução original da imagem do robô (320x240 ou 160x120).
- Melhor rede modifica YOLO para usar ResNet (YOLO v3 adicionou isso) e usa 25% da resolução original da imagem.

# Visão do Humanoide

- Vetor de *features*:

$$x = [t_o \quad t_x \quad t_y \quad t_w \quad t_h]^T$$

- Processamento das *features* segue esquema da YOLO v2:

$$\begin{aligned} p &= \sigma(t_o) \\ x &= c_x + \sigma(t_x) * S_{coord} \\ y &= c_y + \sigma(t_y) * S_{coord} \\ w &= 640 \times 5 \times e^{t_w} \\ h &= 640 \times 5 \times e^{t_h} \end{aligned}$$

- Observação: não usou *confidence*, mas sim probabilidade de presença.

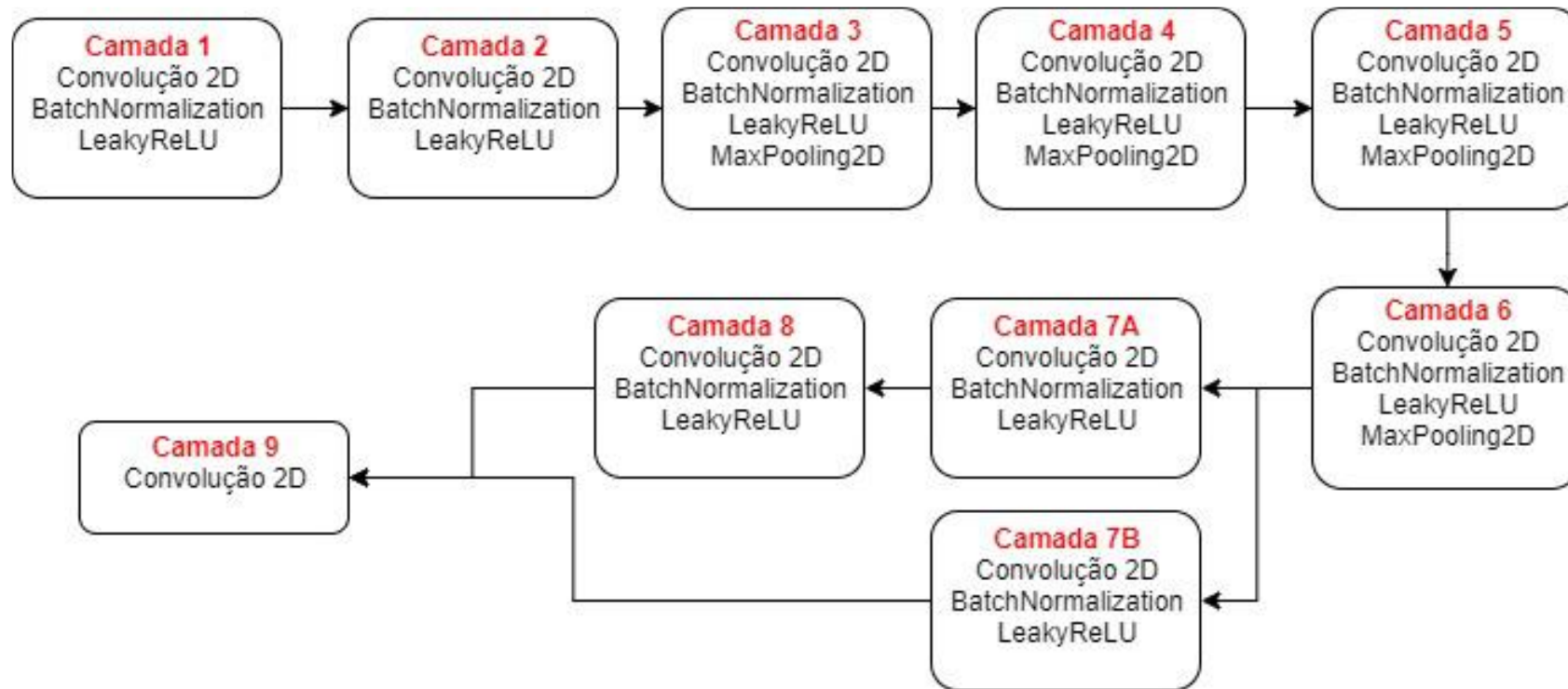
# Visão do Humanoide

- *Loss function:*

$\mathcal{L}$

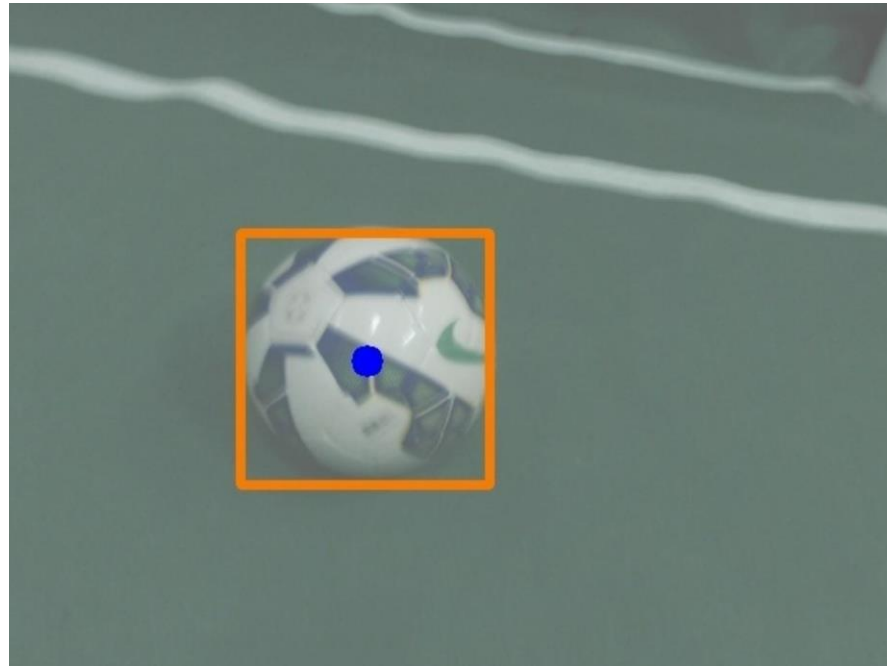
$$\begin{aligned} &= \gamma_{coord} \sum_{i=1}^{S^2} \varepsilon_i^{obj} [(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2] \\ &+ \gamma_{wh} \sum_{i=1}^{S^2} \varepsilon_i^{obj} \left[ (\sqrt{\hat{w}_i} - \sqrt{w_i})^2 \right. \\ &\left. + \left( \sqrt{\hat{h}_i} - \sqrt{h_i} \right)^2 \right] + \gamma_{obj} \sum_{i=1}^{S^2} \varepsilon_i^{obj} (\hat{p}_i - p_i)^2 + \gamma_{noobj} \sum_{i=1}^{S^2} \varepsilon_i^{noobj} (\hat{p}_i - p_i)^2 \end{aligned}$$

# Arquitetura da CNN



# Visão do Humanoide

- *Dataset* construído a partir de imagens do robô.
- Anotação com ferramenta *labelImg* (esforço da equipe).



# Visão do Humanoide

- *Data Augmentation: motion blur, variação de cor e variação de brilho.*
- 12665 imagens no *training set* após *data augmentation*.
- 1490 imagens no *test set*.
- Parâmetros:  $\gamma_{coord} = 5$ ,  $\gamma_{wh} = 5$ ,  $\gamma_{obj} = 1$ ,  $\gamma_{noobj} = 0,5$ .
- Otimizador Adam com parâmetros padrão.
- $\alpha = 0,5 \times 10^{-4}$  fixo.
- Treinamento com 100 épocas.

# Visão do Humanoide

- Inferência no robô usando API de C++ do Tensorflow.
- Compilação do Tensorflow com instruções SSE4.1, SSE4.2, FMA, AVX e AVX2 (melhora desempenho em CPU).
- *Thresholds* de detecção tunados para privilegiar falsos negativos.
- Motivo: visão filtra resultado com filtro de Kalman.
- Assim, falsos positivos são bem piores que falsos negativos.



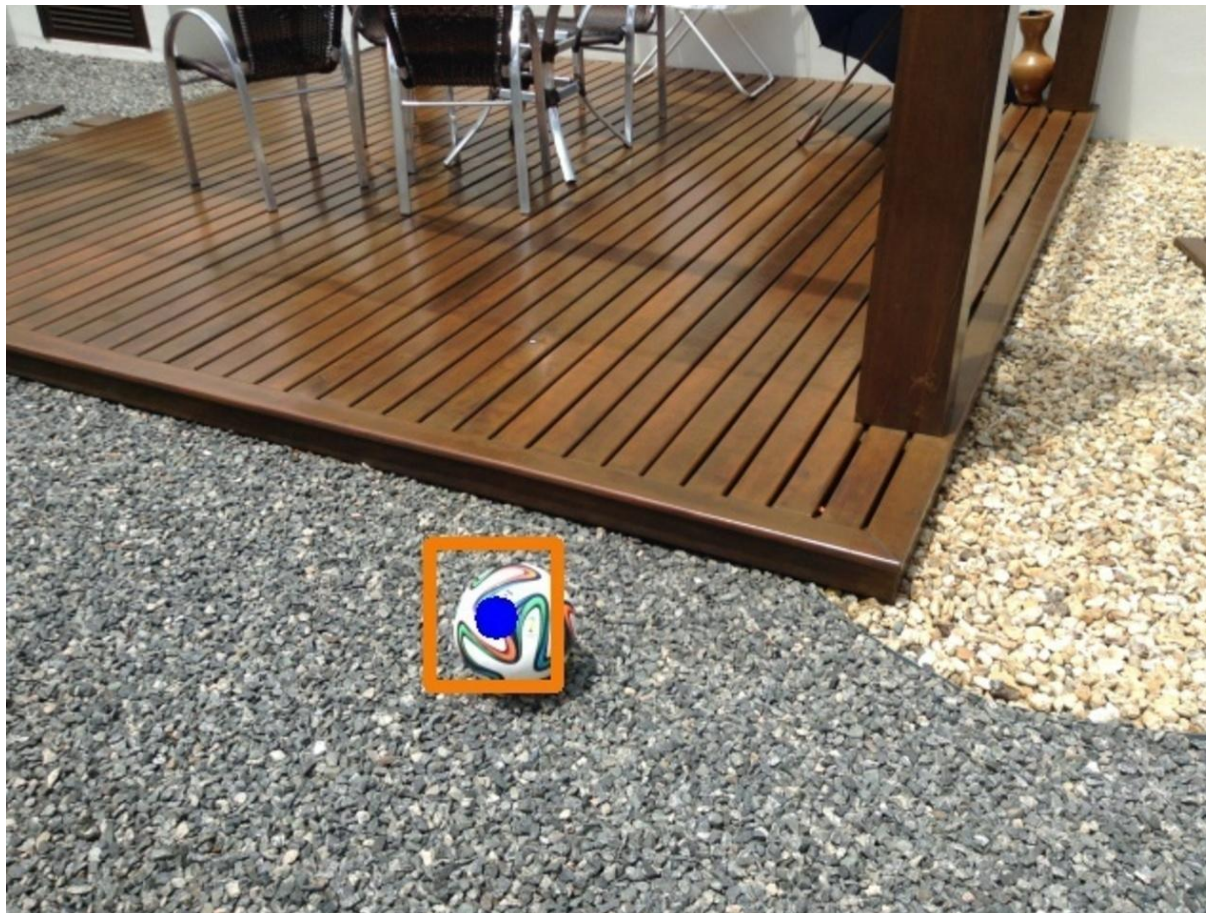
# Visão do Humanoide

- Acurácia de 85,4% no *test set*.
- Tempo de 60-80 ms.
- *IoU* médio: 0,662.
- No começo, nunca achava bola se o chão não fosse verde 😊.
- Problemas com falsos positivos:
  - Encontra bola no ombro dele.
  - De modo geral, adora brilhos metálicos “circulares”.
  - Passamos fita isolante em algumas partes metálicas do robô kkk.
  - Falso positivo na cruz do pênalti (na RoboCup é um círculo!)

# Exemplos de Detecção

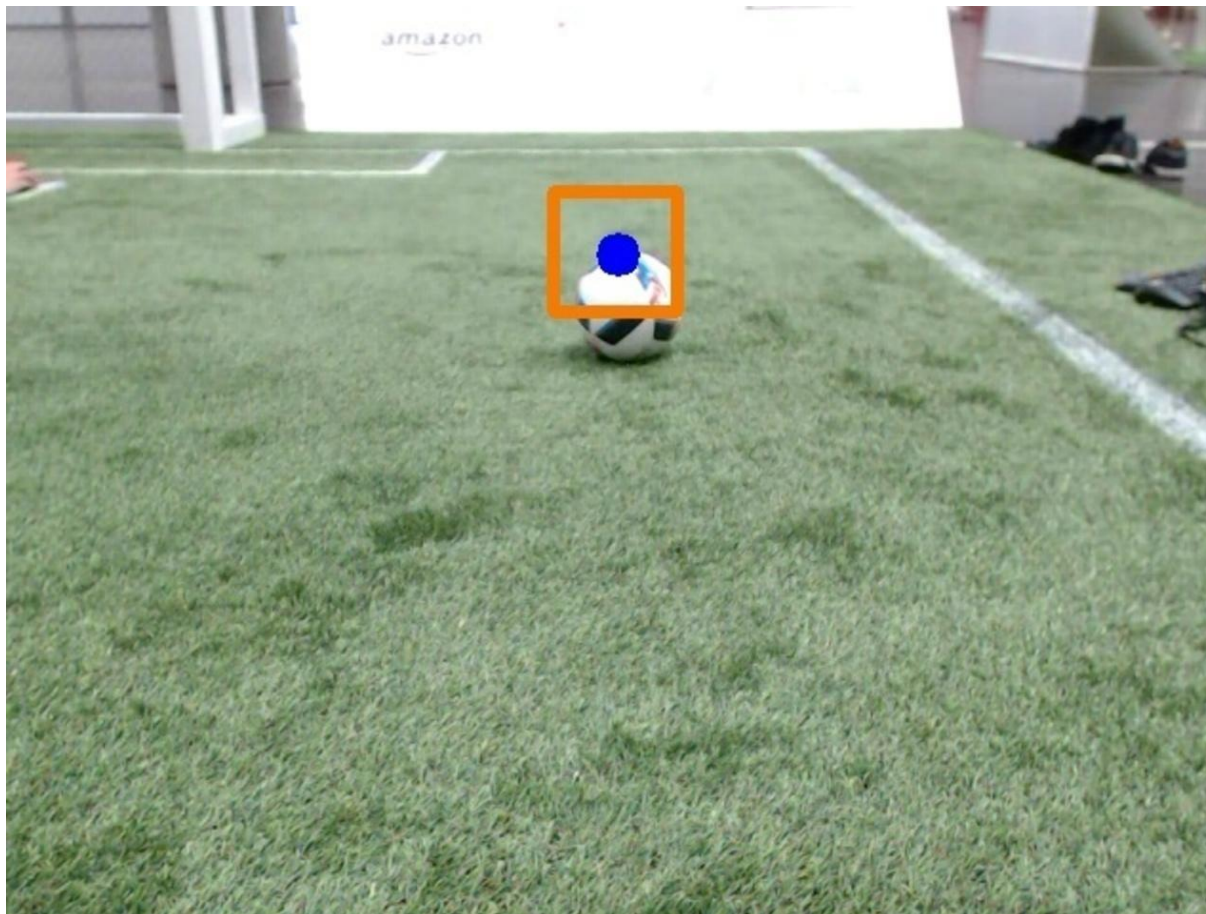


# Exemplos de Detecção





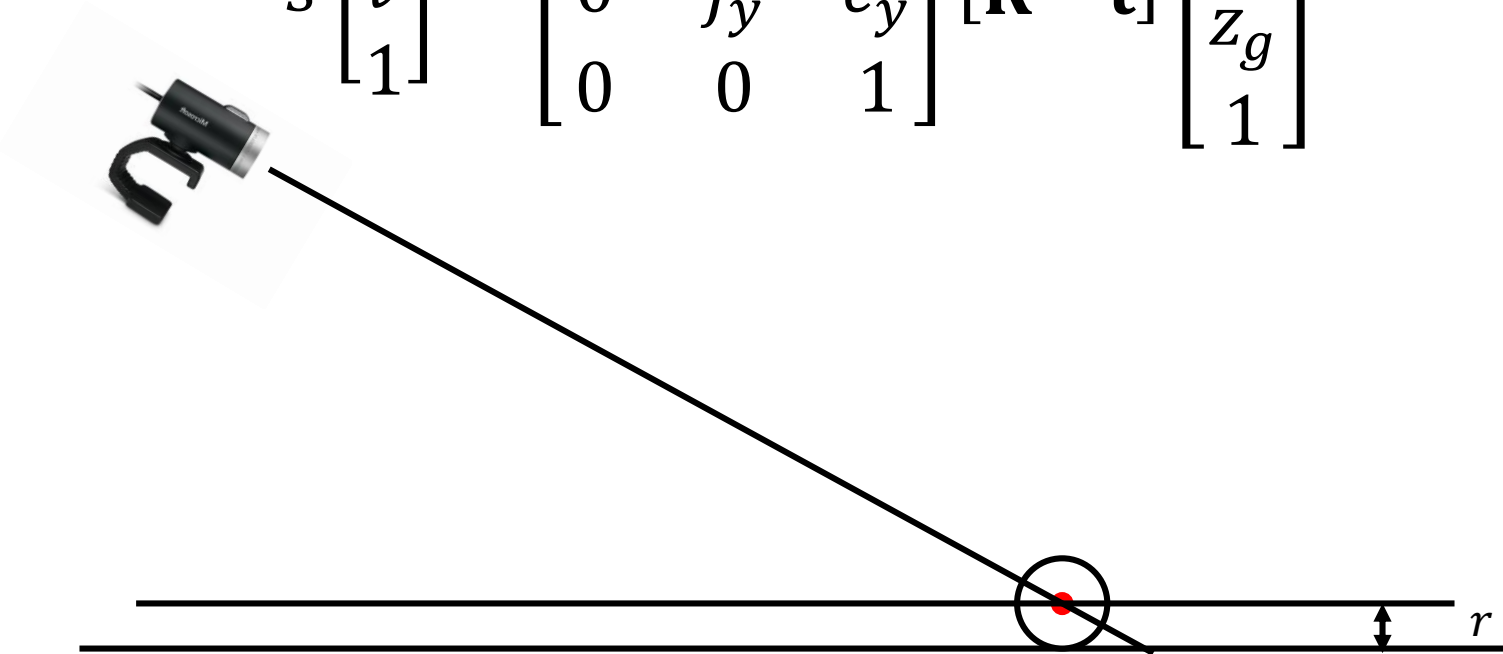
# Exemplos de Detecção



# Visão do Humanoide

- Passando do 2D para o 3D:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R} \quad \mathbf{t}] \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}$$



# Adição de Trave

- Adição de outro *anchor box* com dimensão 2x5.
- Adição de outra *bounding box* no vetor de *features*.
- Testou-se várias possibilidades para representar uma trave.
  - Trave completa.
  - Um lado da trave.
  - Só o ponto de encontro do trave com o chão.
- Empiricamente, o que funcionou melhor foi encontro da trave com o chão.







# Para Saber Mais

- Especialização de Deep Learning do Andrew Ng no Coursera (curso de Convolutional Neural Networks).
- Capítulos 9 do livro: GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning. The MIT Press, 2016.
- Artigos do YOLO:
  - Redmon et al., 2015, You Only Look Once: Unified real-time object detection.
  - Redmon e Farhadi, 2016, YOLO9000: Better, Faster, Stronger.
  - Redmon e Farhadi, 2018, YOLOv3: An Incremental Improvement.

# Laboratório 10

# Laboratório 10

- Implementar YOLO para detecção da bola de futebol de robôs.
- Pensando em como não demorar tanto o treinamento...
- Talvez dar rede pronta e só programar a inferência...