

# Detecção de Múltiplos Objetos em Imagens: Histórico e apresentação dos métodos de Deep Learning mais utilizados

---

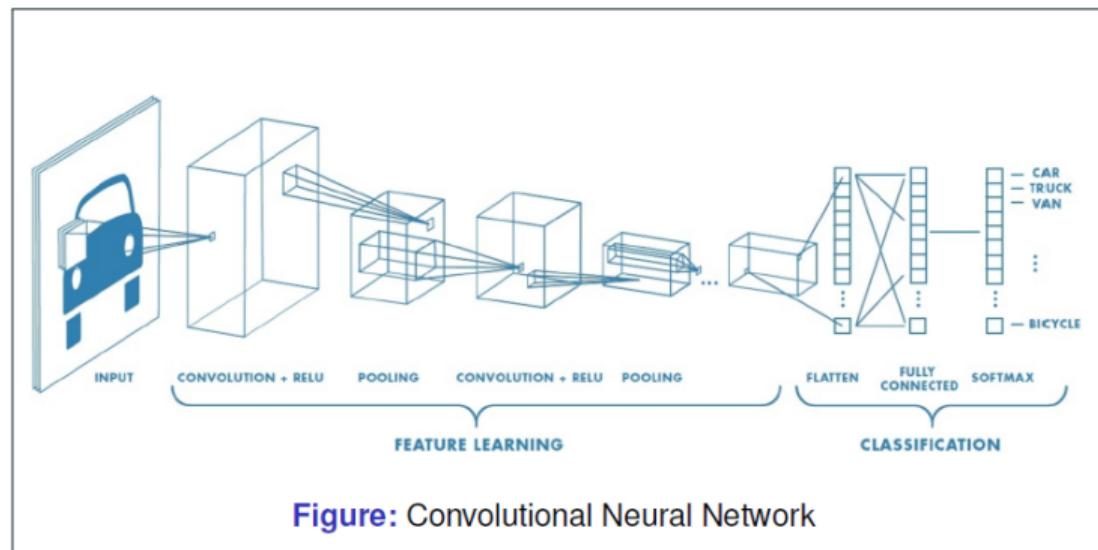


Anderson Andrei Schwertner

15 de Agosto de 2020

PROJETO LSI e DAER

# Recap da apresentação do Prof. Valner

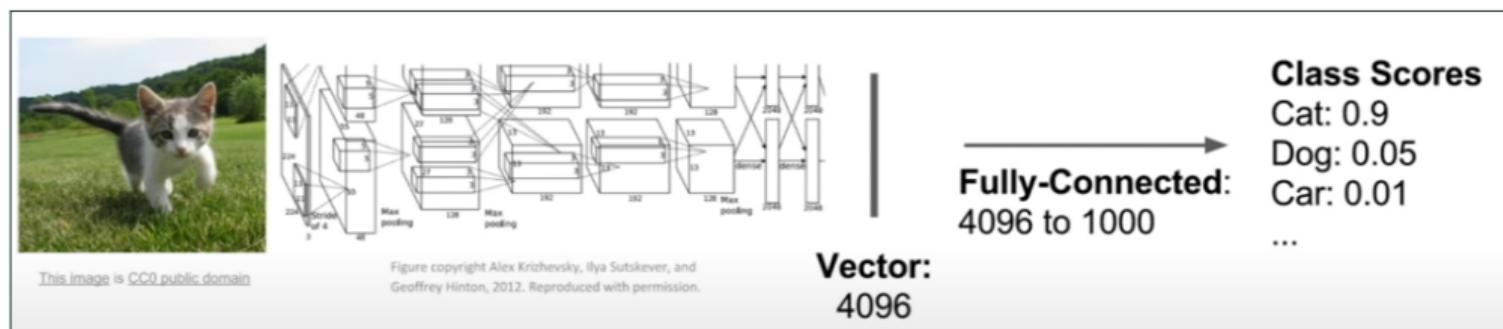


**Figura 1:** Exemplo de arquitetura de uma ConvNet genérica

# Classificação de Imagem: O problema básico

## Classificação de Imagens

A ConvNet só é capaz de responder uma pergunta: **Qual a classe do objeto?**



**Figura 2:** Exemplo da aplicação de classificação em uma imagem

Essa metodologia só nos diz qual objeto está presente na imagem como um todo. Onde está o gato? Quantos gatos existem na imagem? Além do gato, o que há na imagem? São perguntas impossíveis para essa abordagem.

## Limitações atuais

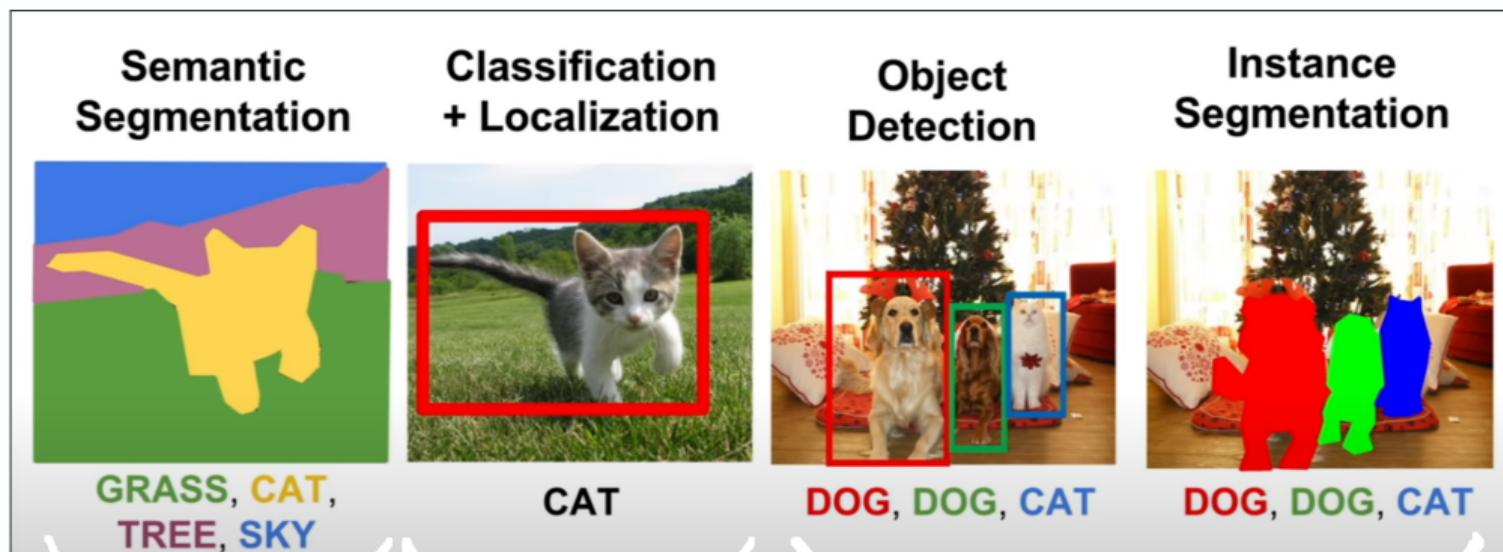
A metodologia de classificação de imagem não é suficiente para nosso problema



**Figura 3:** O que já conseguimos fazer VS O que precisamos fazer

## Algumas das tarefas que utilizam as ConvNets

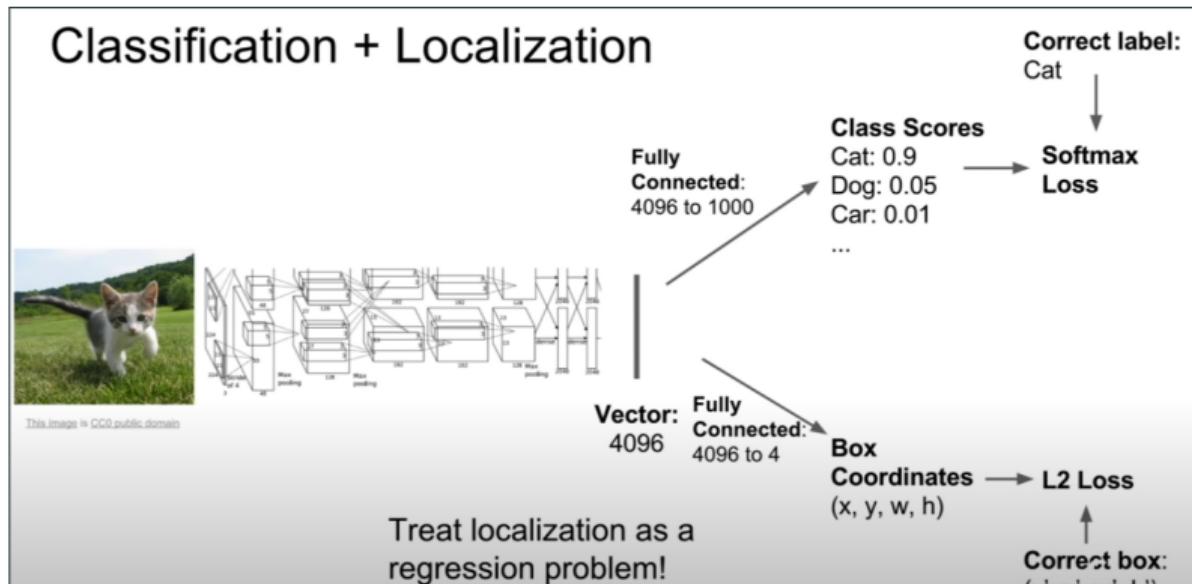
As ConvNets são apenas a base para solucionar cada problema



**Figura 4:** Principais tarefas utilizadas em visão computacional

# Entendendo ConvNets com Múltiplas Habilidades

Como classificar e obter as coordenadas da bouding box de um objeto?



**Figura 5:** Uma alternativa para detectar e localizar um objeto usando ConvNets

## A Ideia Geral

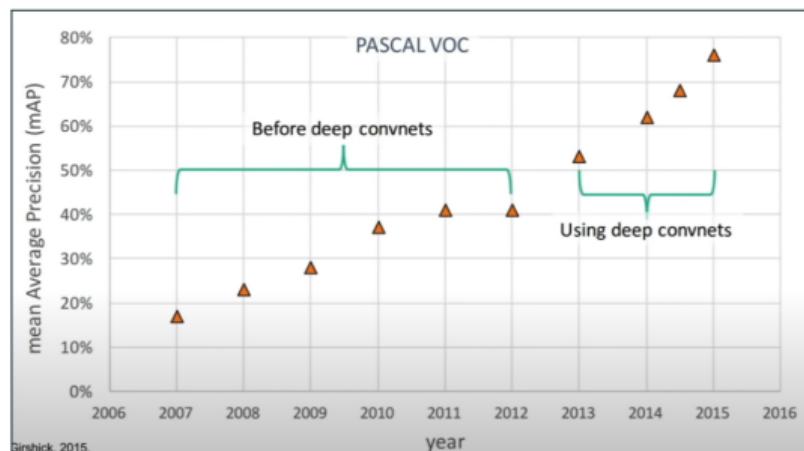
- Começamos definindo um número fixo de classes: gato, cachorro, pássaro, etc.
- Queremos classificar e desenhar uma bouding box cada vez que uma dessas classes aparecer na imagem.
- A diferença para o problema de localização + classificação é que agora não sabemos de antemão quantos objetos existem na imagem.



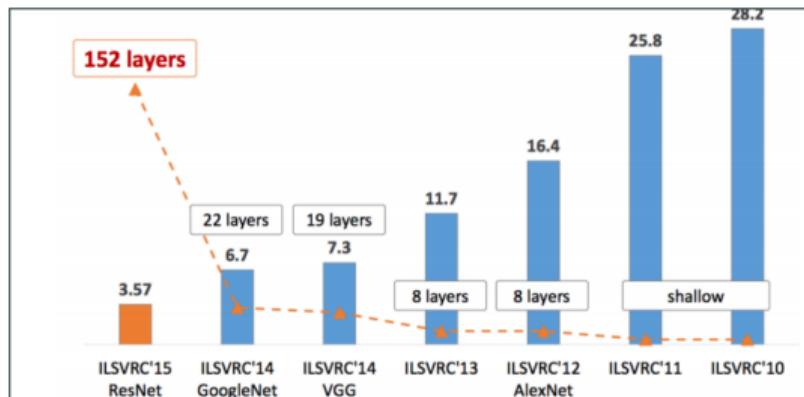
**Figura 6:** Exemplo do resultado de um algoritmo de Detecção de Objetos

# Estado da Arte

## Benefício da utilização de ConvNets assim como em Classificação de Imagens



**Figura 7:** Desempenho dos métodos de Detecção de Objetos ao longo dos anos



**Figura 8:** Desempenho dos métodos de Classificação de Imagens ao longo dos anos

# Exemplificando o problema de múltiplos objetos

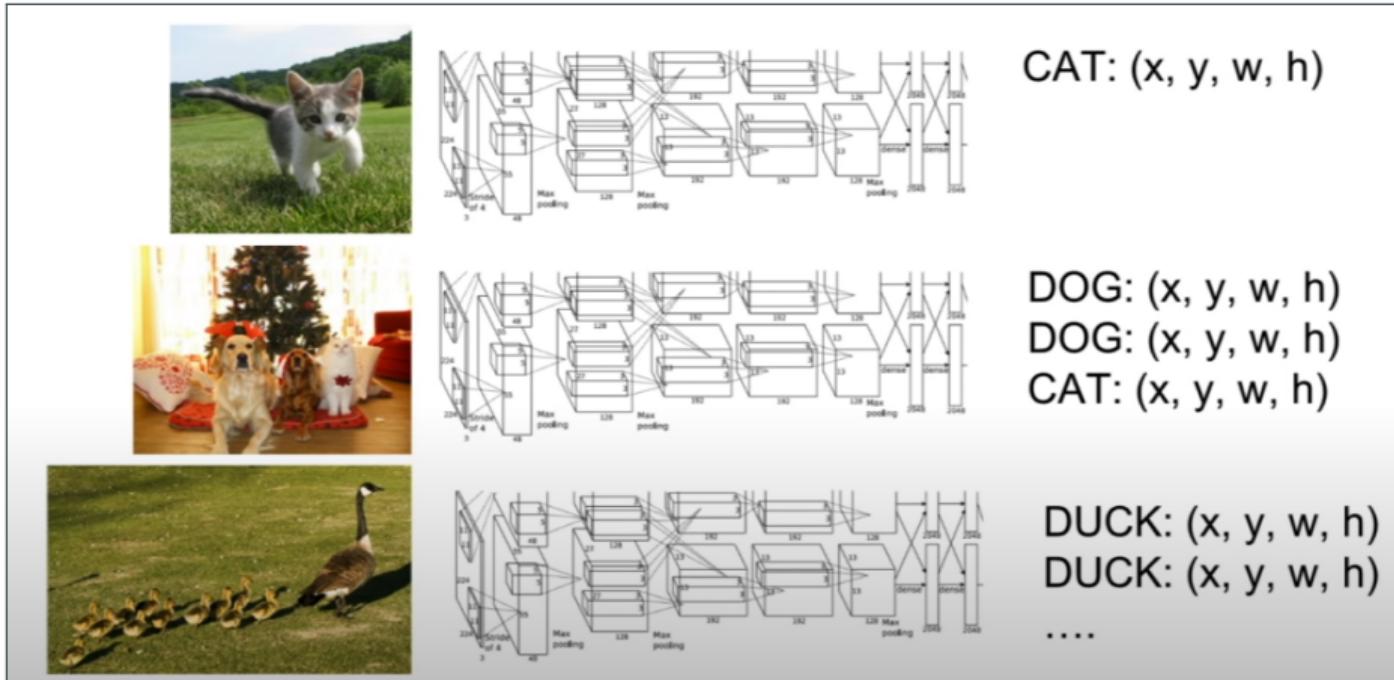
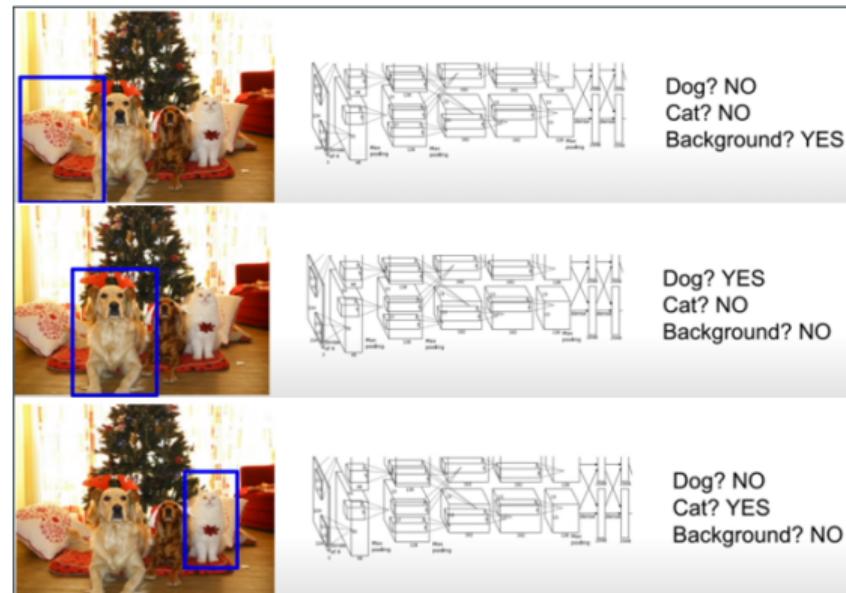


Figura 9: O problema de múltiplos objetos

## Alternativas: Janela Deslizante

- Pode existir qualquer número de objetos, em qualquer localização, com tamanhos diferentes e proporções (aspect ratio) diferentes.
- É fácil de perceber um grande problema: como fazemos para escolher um formato/tamanho/step para a janela?



**Figura 10:** Detectando múltiplos objetos com uma janela deslizante

## Alternativas: Region Proposal

São utilizadas técnicas clássicas de processamento de imagens como detecção de bordas, erosão e análise de histograma para encontrar janelas com potencial de conter um objeto. Um dos algoritmos mais famosos é chamado de **Busca Seletiva** e resulta em 2000 janelas.



**Figura 11:** Exemplo simplificado de Region Proposal

# Algoritmos Práticos: R-CNN (Girshick et al. 2014)

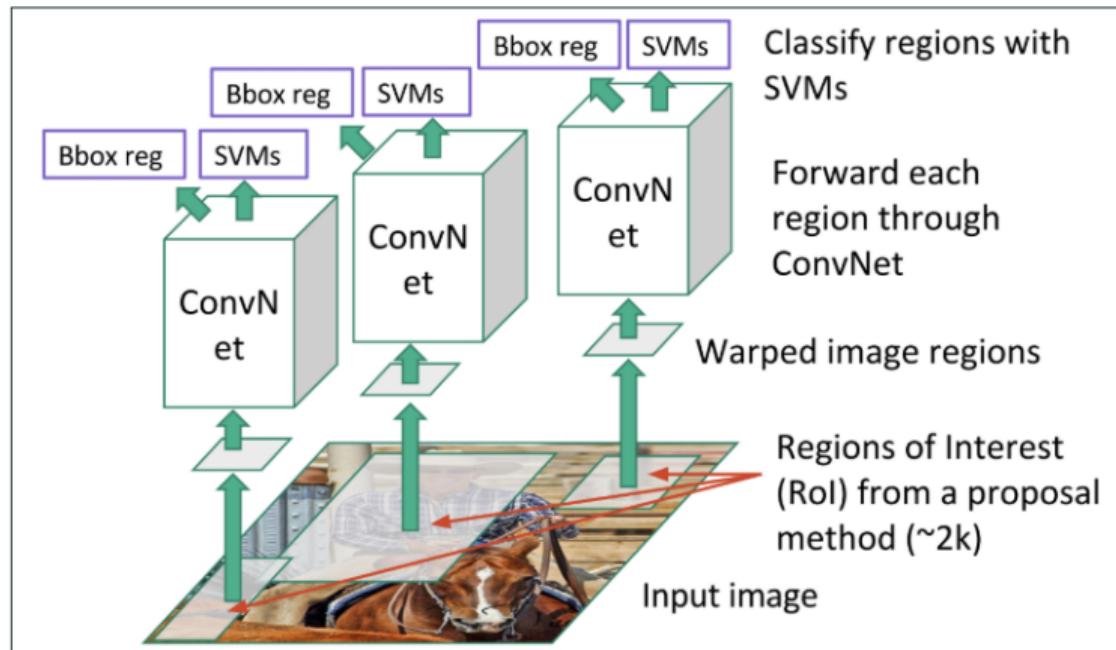


Figura 12: Arquitetura R-CNN

## Algoritmos Práticos: R-CNN (Girshick et al. 2014)

- As regiões de interesse (RoI) são um parâmetro que a ConvNet não aprende. São determinadas por um algoritmo externo.
- Apesar do grande avanço o R-CNN não é essa coca-cola toda. Testar 2000 RoIs é bastante lento e testar uma imagem leva em torno de 47 segundos.
- Percebemos que para treinar um método como esse precisamos de uma base de dados especial, com as coordenadas das bounding boxes de cada objeto.
- Alguns desses problemas foram resolvidos em 2015 com a nova versão do R-CNN.

## Fast R-CNN (Girshick et al. 2015)

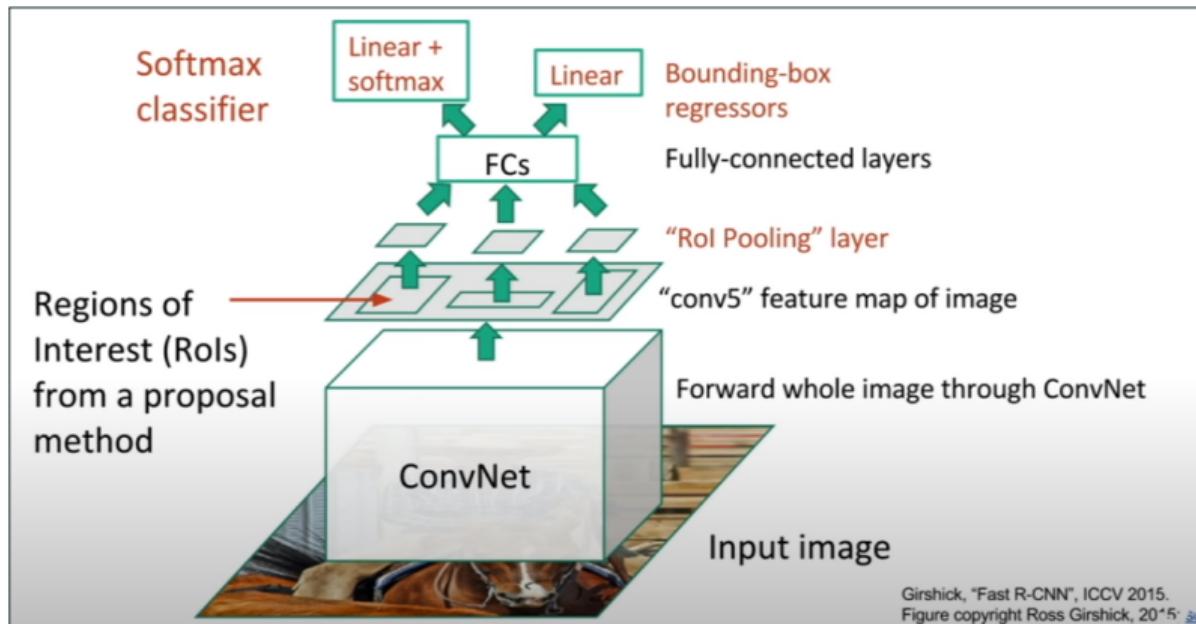
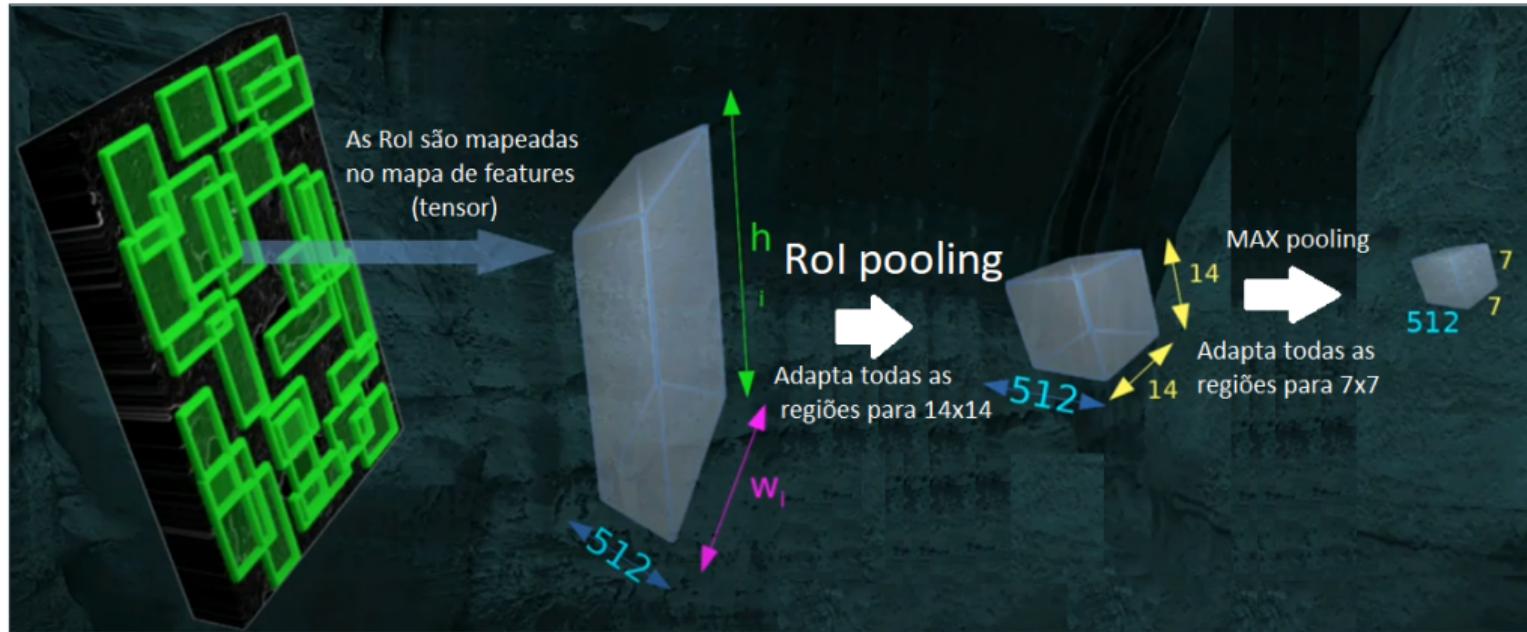


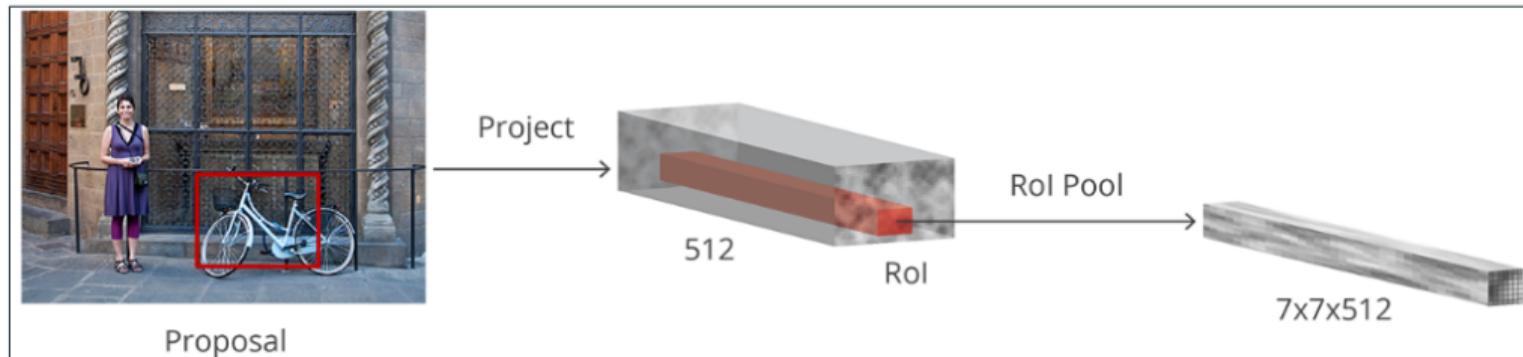
Figura 13: Arquitetura Fast R-CNN

# Rol Pooling: Padronizando o tamanho das regiões



**Figura 14:** Camada de Rol Pooling - Padroniza o tamanho de cada Rol para que possam ser processadas

# Rol Pooling: Padronizando o tamanho das regiões

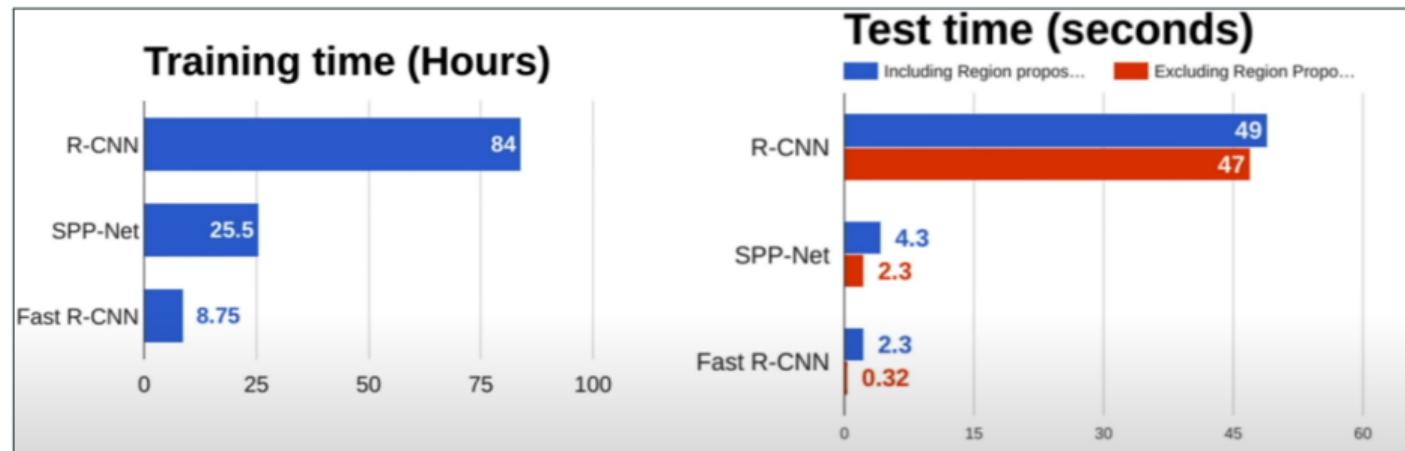


**Figura 15:** Camada de Rol Pooling - Padroniza o tamanho de cada Rol para que possam ser processadas

Cada uma das regiões de interesse são transformadas em um tensor de  $7 \times 7 \times 512$  através de Rol Pooling e Max Pooling. Necessário pois as camadas FC tem tamanho fixo.

## Fast R-CNN (Girshick et al. 2015)

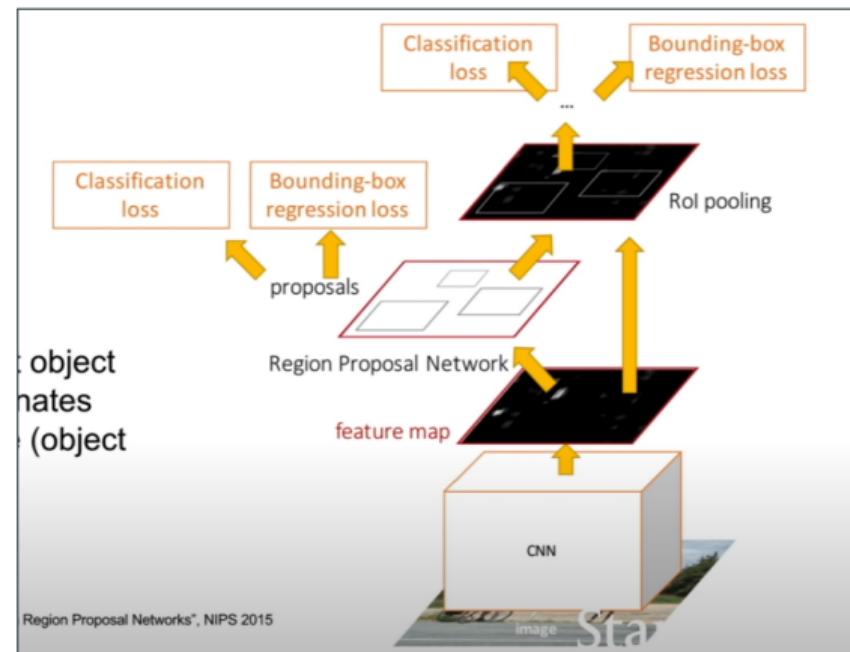
No Fast R-CNN a maior parte do tempo (2 segundos) é gasta no processo de determinar as regiões de interesse através do método de busca seletiva.



**Figura 16:** Comparação do tempo de execução entre os métodos

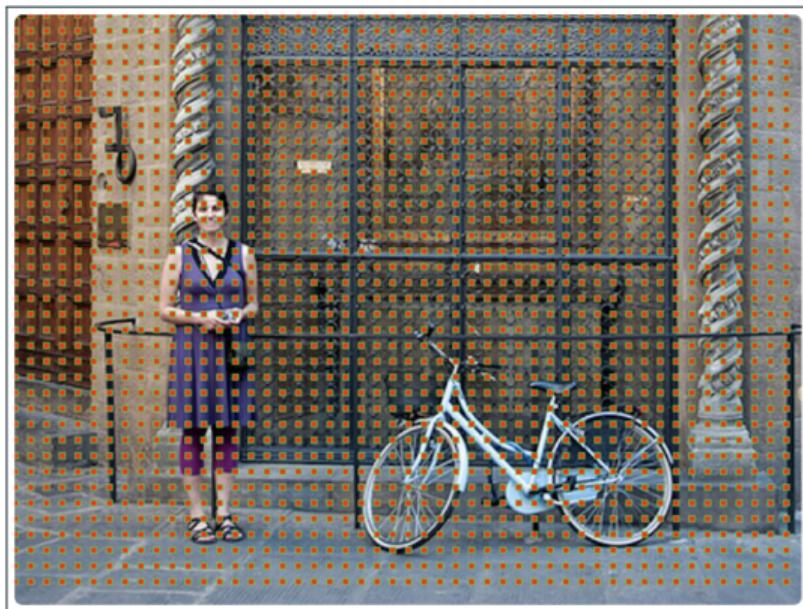
## Faster R-CNN (Girshick et al. 2016)

- A ideia do Faster R-CNN é resolver justamente o problema de que calcular as propostas de região demorava mais que classificar cada uma delas.
- Para solucionar isso, o Faster R-CNN faz com que a própria rede determine as regiões de interesse que provavelmente contém um objeto utilizando uma RPN (Region Proposal Network).



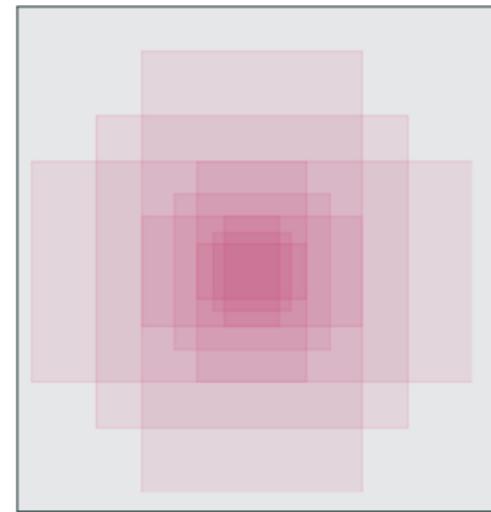
**Figura 17:** Arquitetura Faster R-CNN

# Region Proposal Network (RPN) - Conceito de Âncoras



**Figura 18:** Imagem com o centro das âncoras

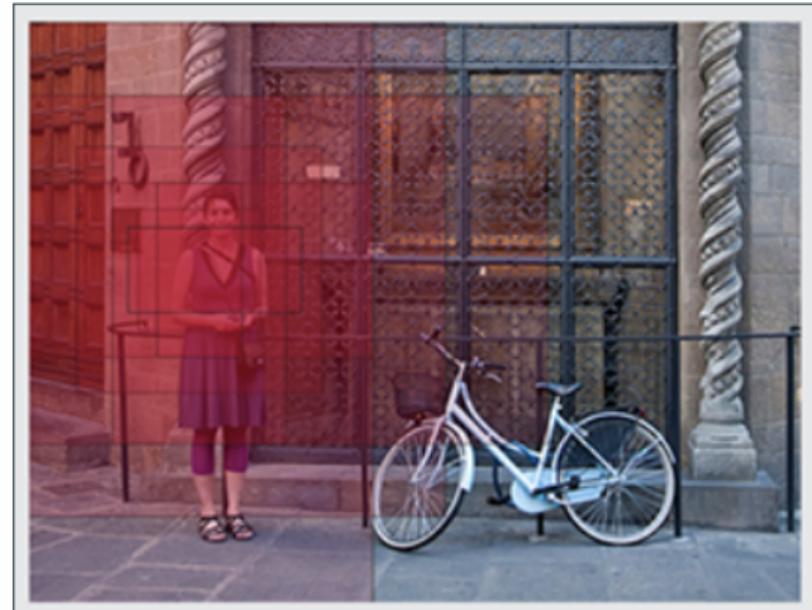
Anderson Andrei Schwertner



**Figura 19:** As âncoras tem diferentes tamanhos e aspect ratios e são aplicadas em cada um dos pontos da imagem

## Region Proposal Network (RPN) - Conceito de Âncoras

- O conjunto de âncoras é aplicado em cada ponto da imagem.
- O exemplo exibe uma imagem apenas com intuito ilustrativo, na verdade, as âncoras são posicionadas sob o mapa de features.



**Figura 20:** Âncoras posicionadas sob um dos pontos da imagem

# Region Proposal Network (RPN) - Avaliando cada Âncora

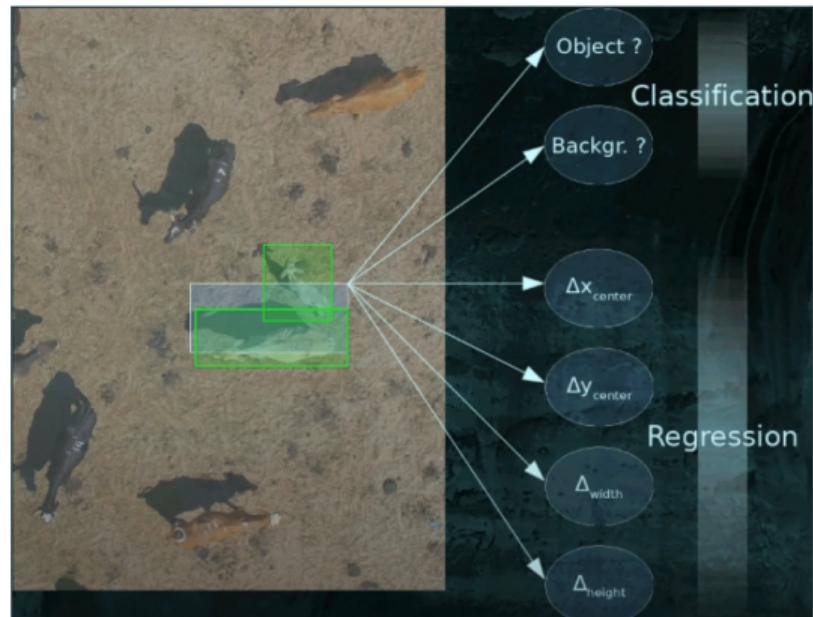
Para cada âncora temos duas saídas de classificação:

- Probabilidade de ser um objeto.
- Probabilidade de ser background.

E quatro saídas de regressão:

- Ajuste da posição em X e Y.
- Ajuste de altura e largura.

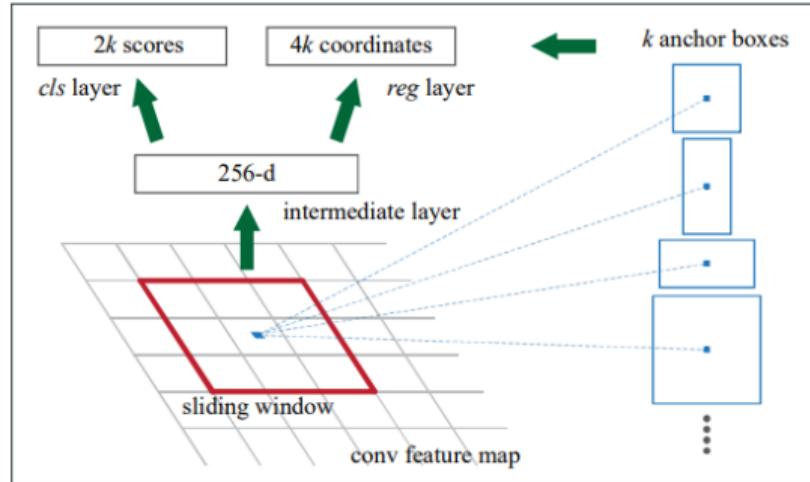
A RPN pega todas as âncoras, testa e retorna um conjunto de boas propostas para objetos.



**Figura 21:** Saídas da RPN para cada Âncora - Classificação e Regressão

# Region Proposal Network (RPN): Convolução + Janela Deslizante

- Uma âncora é centralizada na janela deslizante em questão e está associada a uma escala e proporção. Por padrão, usamos 3 escalas e 3 proporções, produzindo  $k = 9$  âncoras em cada posição de deslizamento.
- Considerando um mapa de features com tamanho 2400 temos 21600 âncoras no total.



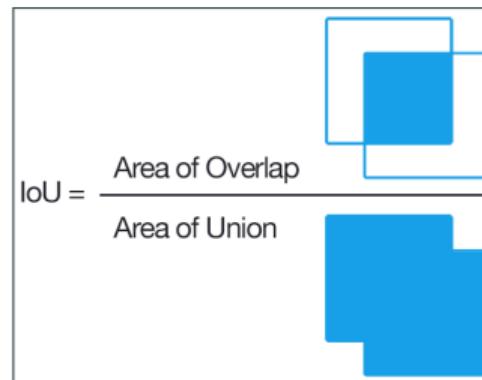
**Figura 22:** Obtenção das âncoras na RPN - Convolução e Janela Deslizante

## Region Proposal Network (RPN): Seleção das Melhores Âncoras



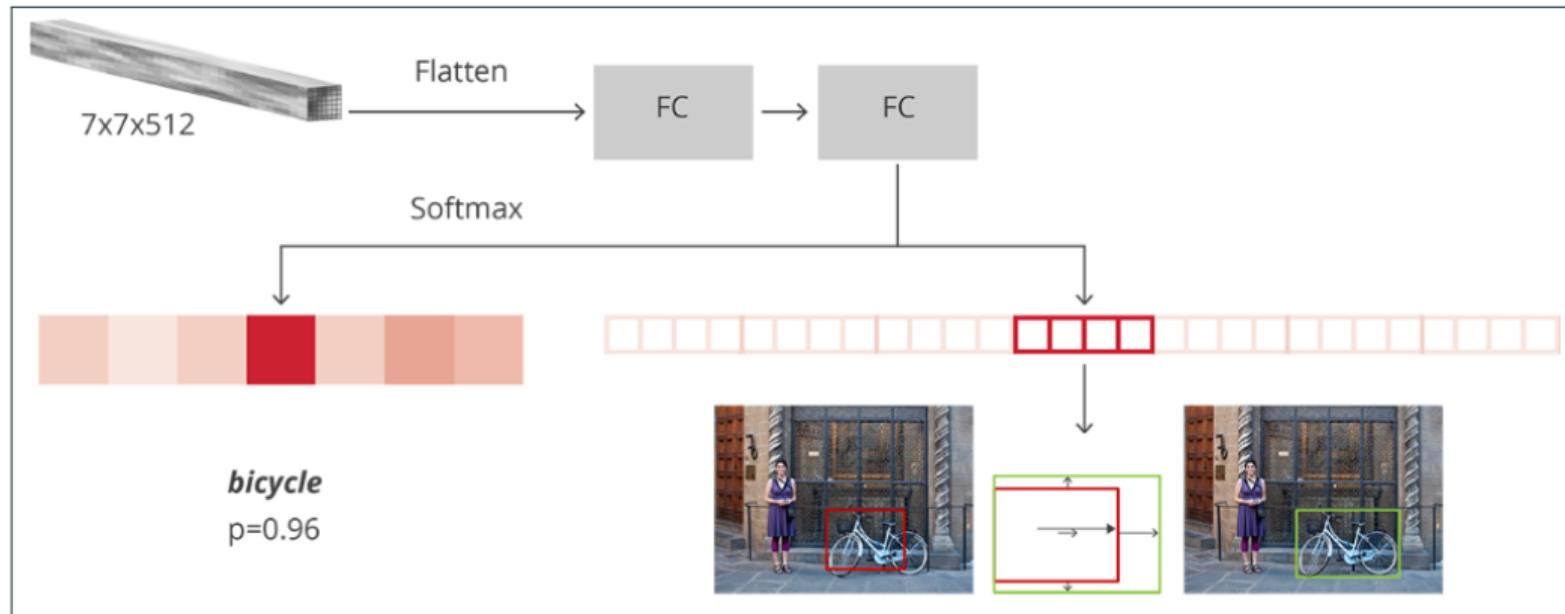
**Figura 23:** Exemplo de overlap das âncoras  
Anderson Andrei Schwertner

São descartadas as propostas que tem  $\text{IoU} \geq 0,7$  com uma proposta com probabilidade de objeto mais alta.



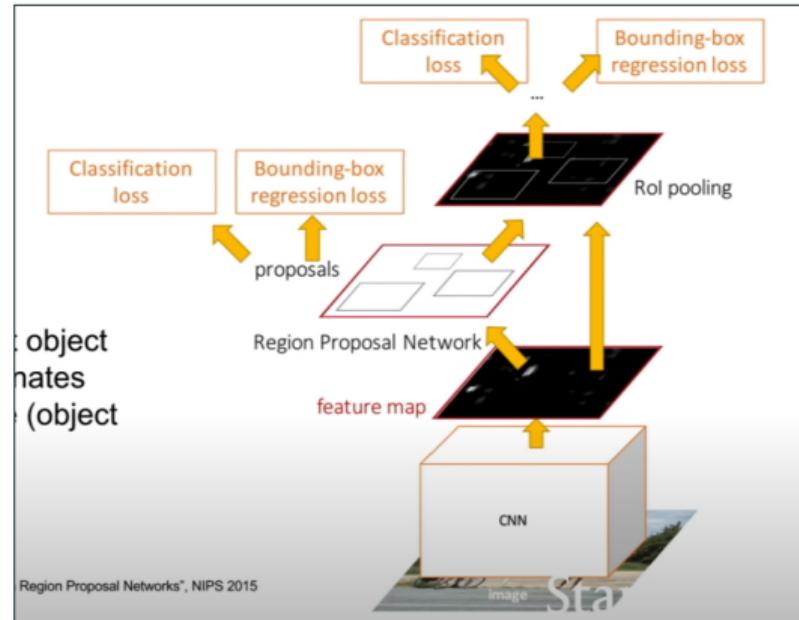
**Figura 24:** Cálculo da métrica IoU (Intersection over Union)

# Faster R-CNN: Classificação e Bounding Boxes



**Figura 25:** Arquitetura simplificada das camadas FC: Classificação e correção das coordenadas da bouding box

## Even Faster R-CNN (Girshick et al. 2017)



**Figura 26:** Slide só de sacanagem mesmo, não existe o Even Faster R-CNN

# YOLO - You Only Look Once (Redmon et al. 2015)

Bounding Boxes têm um nível de confiança exemplificado aqui pela espessura da linha



**Figura 27:** Divisão da imagem em uma grade SxS ( $S=7$ )

Anderson Andrei Schwertner



**Figura 28:** Cada célula é responsável por prever B bounding boxes



**Figura 29:** Previsão de bounding boxes em uma célula

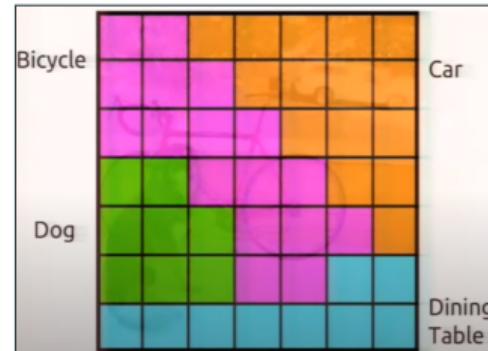
# YOLO - You Only Look Once (Redmon et al. 2015)

Para cada célula é assignada uma classe: Se existir um objeto ali, ele é da classe X.

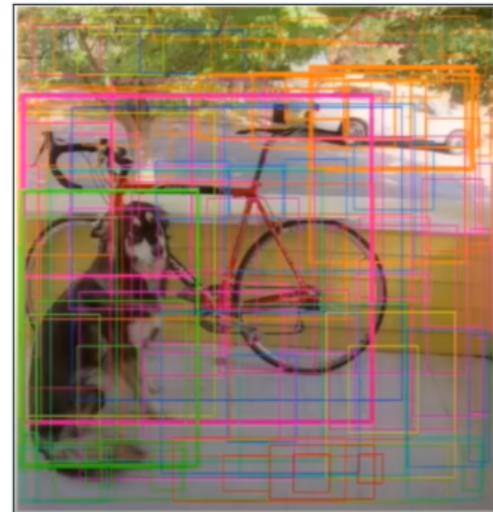


**Figura 30:** Representação de todas as bounding boxes

Anderson Andrei Schwertner



**Figura 31:** Previsão de classe:  
Probabilidade condicional



**Figura 32:** Combinação:  
*class-specific confidence* por  
bounding box

# YOLO - You Only Look Once (Redmon et al. 2015)

- Aplica-se um threshold nas bouding boxes para eliminar as com baixa confiança.
- Aplica-se Non-Maximum Supression para eliminar overlap.

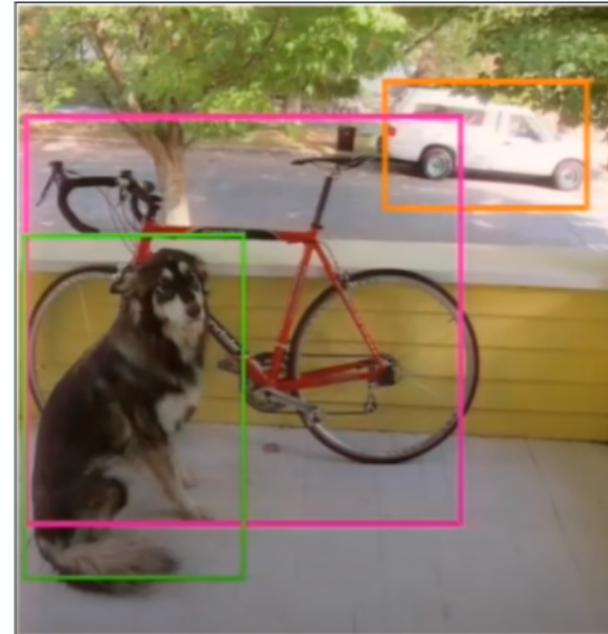


Figura 33: Resultado final do processo

# YOLO - You Only Look Once (Redmon et al. 2015)

YOLO utiliza um framework chamado DarkNet-13 inspirado na GoogLeNet. Foi originalmente treinado para 20 classes de objetos (Base PASCAL VOC.)

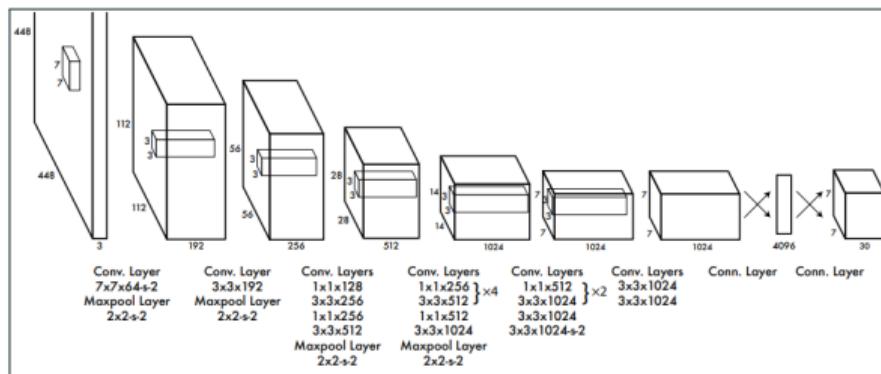


Figura 34: Arquitetura DarkNet

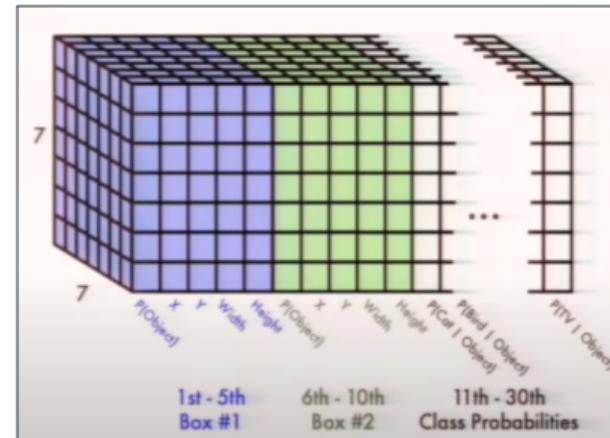


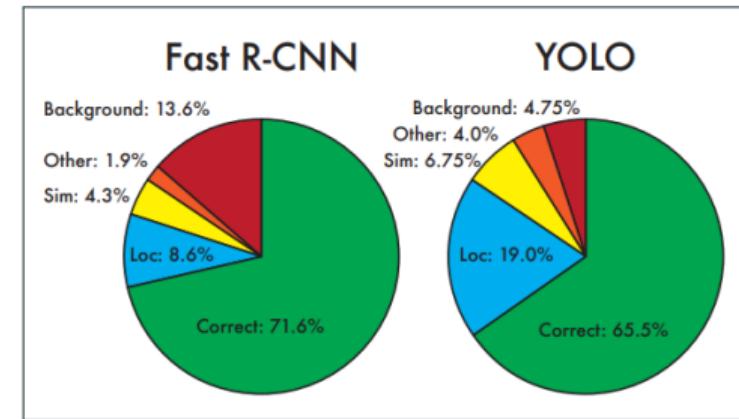
Figura 35: Tensor de saída

# YOLO - You Only Look Once (Redmon et al. 2015)

Apesar de mais rápido, o YOLO é menos preciso que o Faster R-CNN.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

**Figura 36:** Desempenho e velocidade de alguns métodos



**Figura 37:** Comparação de classificação entre YOLO e Fast R-CNN

# YOLOv2 - Better, Faster, Stronger (Redmon & Farhadi 2016)

Alguns dos upgrades da segunda versão:

- Nova CNN: DarkNet-19 com 19 camadas.
- Rede treinada em resolução superior: 448x448 vs 256x256 na primeira versão.
- Utilização de Âncoras (Dimension Clusters).
- Treinamento multi-resolução.
- Aumento no tamanho do mapa de features, de 7x7 para 13x13.

	YOLO	YOLOv2							
batch norm?	✓	✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?	✓	✓	✓	✓	✓	✓	✓	✓	✓
convolutional?	✓	✓	✓	✓	✓	✓	✓	✓	✓
anchor boxes?	✓	✓	✓	✓	✓	✓	✓	✓	✓
new network?	✓	✓	✓	✓	✓	✓	✓	✓	✓
dimension priors?		✓	✓	✓	✓	✓	✓	✓	✓
location prediction?			✓	✓	✓	✓	✓	✓	✓
passthrough?				✓	✓	✓	✓	✓	✓
multi-scale?					✓	✓	✓	✓	✓
hi-res detector?						✓	✓	✓	✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	78.6

Figura 38: Upgrades da segunda versão do Yolo

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	<b>78.6</b>	40

Figura 39: Comparação entre métodos estado da arte

# YOLOv3 - (Redmon & Farhadi 2016)

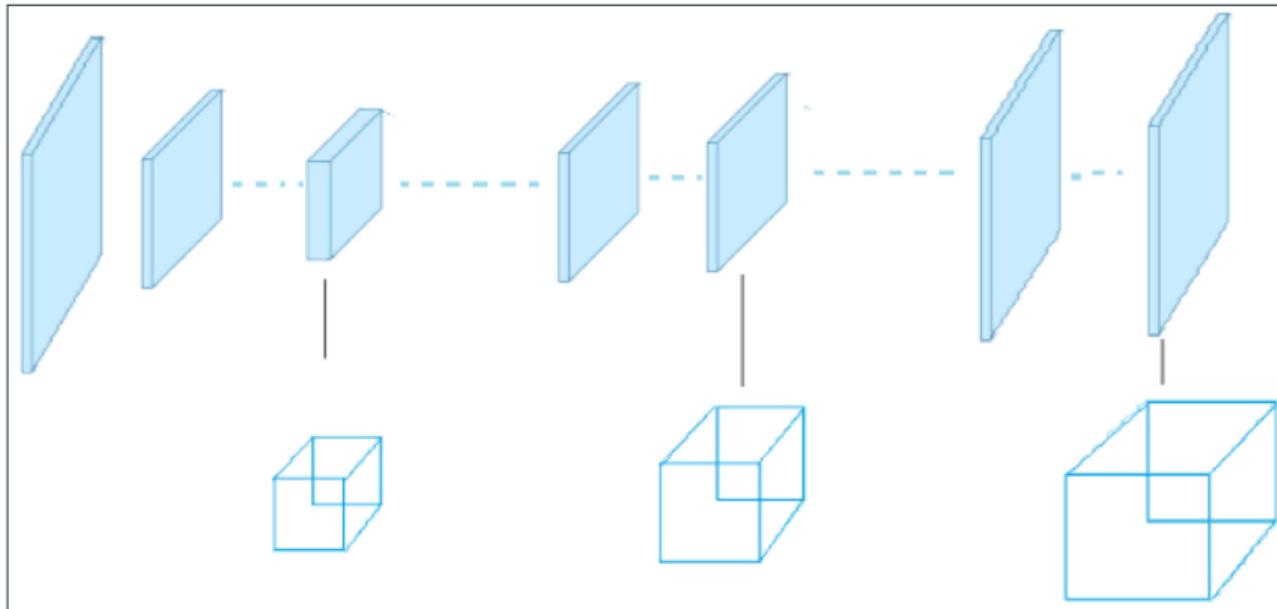
Upgrades da terceira versão:

- Agora as âncoras tem um score que representa a probabilidade de ser um objeto.
- Remove overlap com Non-Maximum Supression.
- Cada âncora também prevê a classe do objeto.
- YOLOv3 prevê 3 bounding boxes em 3 estágios diferentes da rede.
- Nova CNN: DarkNet-53. Como o nome sugere, possui 53 camadas.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	<b>171</b>
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	<b>77.6</b>	<b>93.8</b>	29.4	1090	37
Darknet-53	77.2	<b>93.8</b>	18.7	<b>1457</b>	78

**Figura 40:** Comparação entre CNNs

# YOLOv3 - (Redmon & Farhadi 2016)



**Figura 41:** Detecção de objetos em 3 estágios diferentes da CNN

# YOLOv3 - (Redmon & Farhadi 2016)

- YOLOv3 tem os melhores resultados na base COCO.
- O único método com melhor desempenho nas métricas de precisão, RetinaNet, tem um tempo de processamento 3,8X maior.

Average Precision (AP):	
AP	% AP at IoU=.50:.05:.95 (primary challenge metric)
AP <sub>IoU=.50</sub>	% AP at IoU=.50 (PASCAL VOC metric)
AP <sub>IoU=.75</sub>	% AP at IoU=.75 (strict metric)
AP Across Scales:	
AP <sub>small</sub>	% AP for small objects: area < 32 <sup>2</sup>
AP <sub>medium</sub>	% AP for medium objects: 32 <sup>2</sup> < area < 96 <sup>2</sup>
AP <sub>large</sub>	% AP for large objects: area > 96 <sup>2</sup>
Average Recall (AR):	
AR <sub>max=1</sub>	% AR given 1 detection per image
AR <sub>max=10</sub>	% AR given 10 detections per image
AR <sub>max=100</sub>	% AR given 100 detections per image
AR Across Scales:	
AR <sub>small</sub>	% AR for small objects: area < 32 <sup>2</sup>
AR <sub>medium</sub>	% AR for medium objects: 32 <sup>2</sup> < area < 96 <sup>2</sup>
AR <sub>large</sub>	% AR for large objects: area > 96 <sup>2</sup>

Figura 42: Métricas utilizadas na base COCO

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	<b>52.1</b>
<i>One-stage methods</i>							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	<b>40.8</b>	<b>61.1</b>	<b>44.1</b>	<b>24.1</b>	<b>44.2</b>	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figura 43: Comparação entre métodos na base COCO

## YOLOv3 - (Redmon &amp; Farhadi 2016)

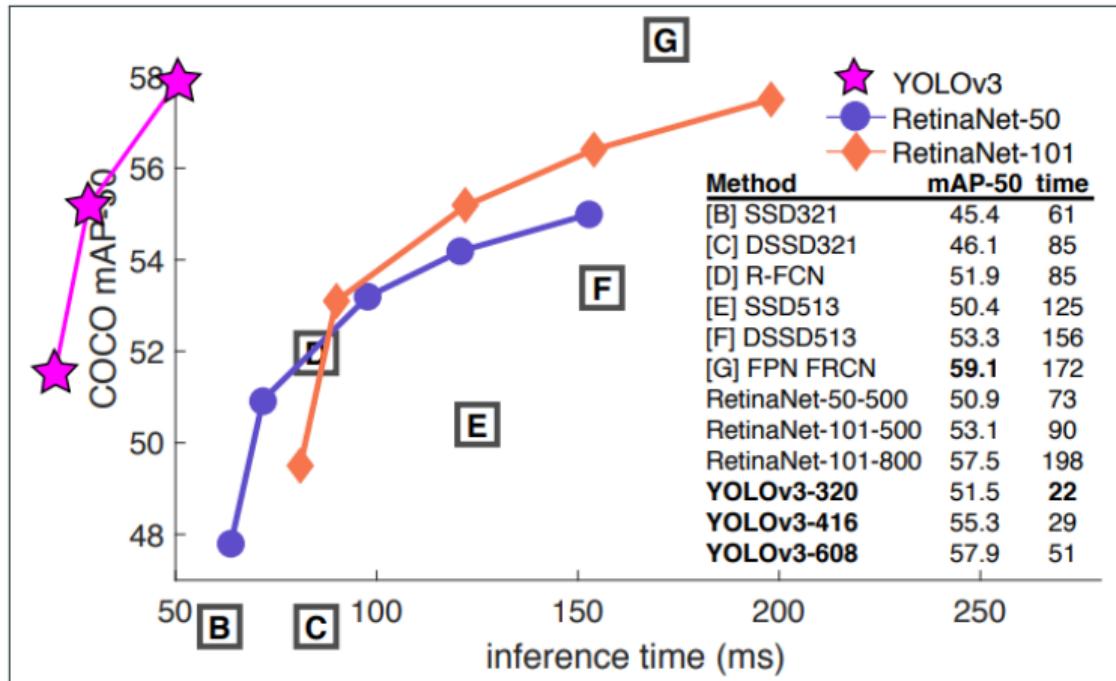
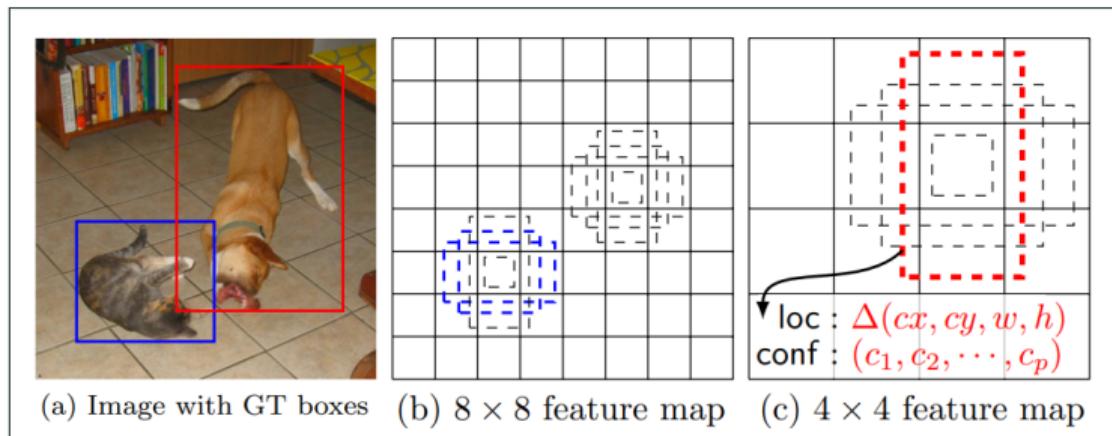


Figura 44: Desempenho do YOLOv3 (gráfico com origem deslocada nos dois eixos)

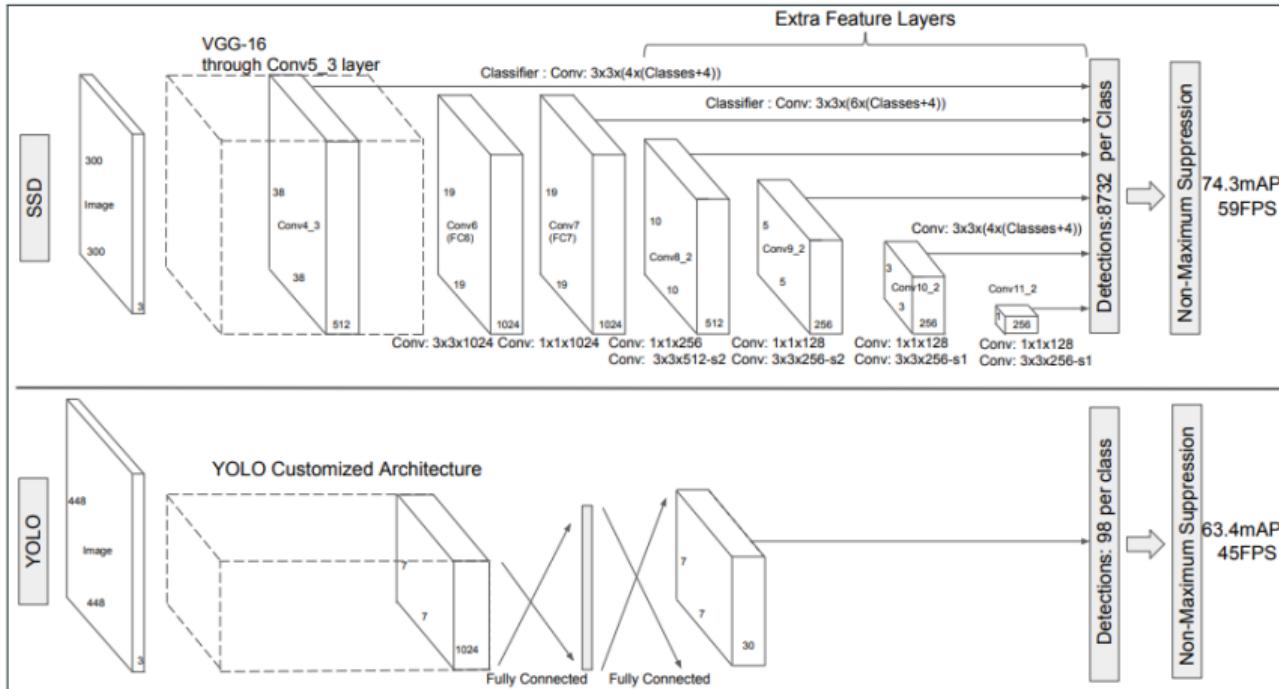
# Single Shot Detector (Liu et al. 2016)

- O SSD é a inspiração para a detecção em vários estágios utilizada no YOLOv3.
- São utilizadas as âncoras (6) do Fast R-CNN em 3 estágios diferentes da rede.
- A aplicação em múltiplas resoluções melhora a discretização da imagem.



**Figura 45:** Âncoras aplicadas em escalas diferentes: O campo de visão da âncora é o mesmo em **b** e **c**

# Single Shot Detector (Liu et al. 2016)



**Figura 46:** Arquitetura SSD vs YOLO

# Single Shot Detector (Liu et al. 2016)

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ $1000 \times 600$
Fast YOLO	52.7	155	1	98	$448 \times 448$
YOLO (VGG16)	66.4	21	1	98	$448 \times 448$
SSD300	74.3	46	1	8732	$300 \times 300$
SSD512	76.8	19	1	24564	$512 \times 512$
SSD300	74.3	59	8	8732	$300 \times 300$
SSD512	76.8	22	8	24564	$512 \times 512$

**Figura 47:** Desempenho na base PASCAL VOC

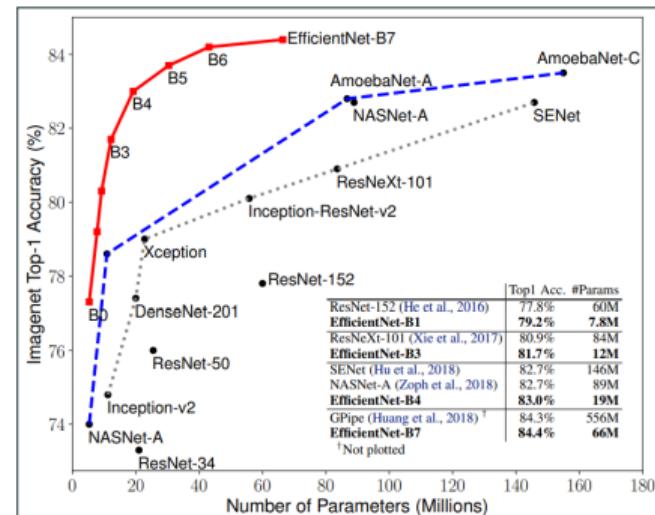
Method	data	Avg. Precision, IoU: 0.5:0.95 0.5 0.75			Avg. Precision, Area: S M L			Avg. Recall, #Dets: 1 10 100			Avg. Recall, Area: S M L		
		19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast [6]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Fast [24]	train	21.9	42.7	-	-	-	-	-	-	-	-	-	-
Faster [2]	trainval	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
ION [24]	train	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
Faster [25]	trainval	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD300	trainval35k	26.8	46.5	27.8	9.0	28.9	41.9	24.8	37.5	39.8	14.0	43.5	59.0
SSD512	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5

**Figura 48:** Desempenho na base COCO

## Ideia geral

Precisamos estudar a fundo e definir a melhor combinação de ConvNet + Object Detector para nossa aplicação. Podemos usar, por exemplo:

- VGG16 + Faster R-CNN;
- MobileNet + SSD;
- Darknet19 + YOLOv3;
- Inception + SSD;
- EfficientNet + FPN;
- ResNet + RetinaNet;



**Figura 49:** Nova state-of-the-art ConvNet chamada EfficientNet, Novembro de 2019

## Considerações Finais

- Apresentação, imagens, papers e afins em <https://tinyurl.com/v2qlro>
- EfficientNet+YOLOv3 <https://www.youtube.com/watch?v=b5EIO22oN0E>
- Verifiquem o canal dele para mais vídeos testando diversas combinações em várias GPUs

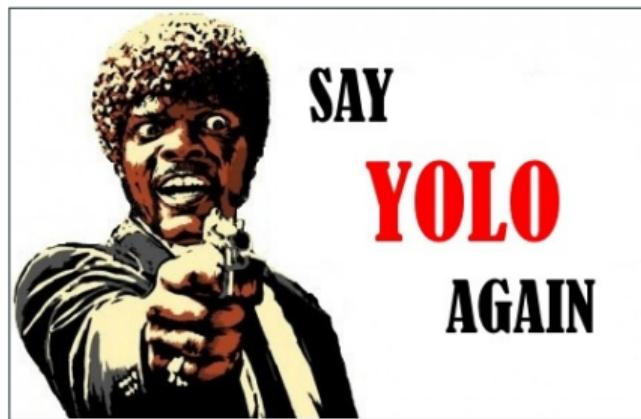


Figura 50: YOLO

# Detecção de Múltiplos Objetos em Imagens: Histórico e apresentação dos métodos de Deep Learning mais utilizados

---



Anderson Andrei Schwertner

15 de Agosto de 2020

PROJETO LSI e DAER