[PSI3472 aula 11, Início]

Detecção de objetos

1. Introdução [https://d2l.ai/chapter_computer-vision/bounding-box.html]

Nesta aula, vamos estudar a detecção de objetos usando CNNs. A classificação de imagem recebe uma imagem e gera um rótulo (classificação) da imagem. A detecção de objetos recebe uma imagem e gera um conjunto de caixas delimitadoras (*bounding boxes*), juntamente com as classificações e as notas de confiança (figura 1).

Figura 1: Exemplos de detecções de objetos.

[PSI3472 aula 11, pular a partir daqui]

2. Avaliação de um modelo para detecção de objetos.

Na classificação de imagens, são utilizadas métricas de erro como taxa de erro, acuracidade, sensibilidade, especificidade, ROC-AUC, etc. Estas métricas não podem ser usadas na detecção de objetos. Precisamos de outras métricas para medir a qualidade da detecção de objetos.

2.1 Intersecção sobre União (IoU) [https://en.wikipedia.org/wiki/Jaccard_index]

Normalmente, detecção de objetos trabalha com caixas delimitadoras (bounding boxes). Uma caixa delimitadora é normalmente especificada como (x, y, w, h), onde (x, y) são as coordenadas da caixa (pode ser o centro da caixa ou o seu canto superior esquerdo) e (w, h) são a largura e altura da caixa.

Nota: A unidade desses valores podem estar em pixels ou em frações da largura/altura da imagem.

Interseção sobre União (IoU), também conhecido como Índice de Jaccard, é calculada dividindo a área da intersecção das duas caixas pela área da união delas. Em termos matemáticos, IoU é:

$$IoU = \frac{(\text{área da intersecção})}{(\text{área da união})} = \frac{|A \cap B|}{|A \cup B|}$$

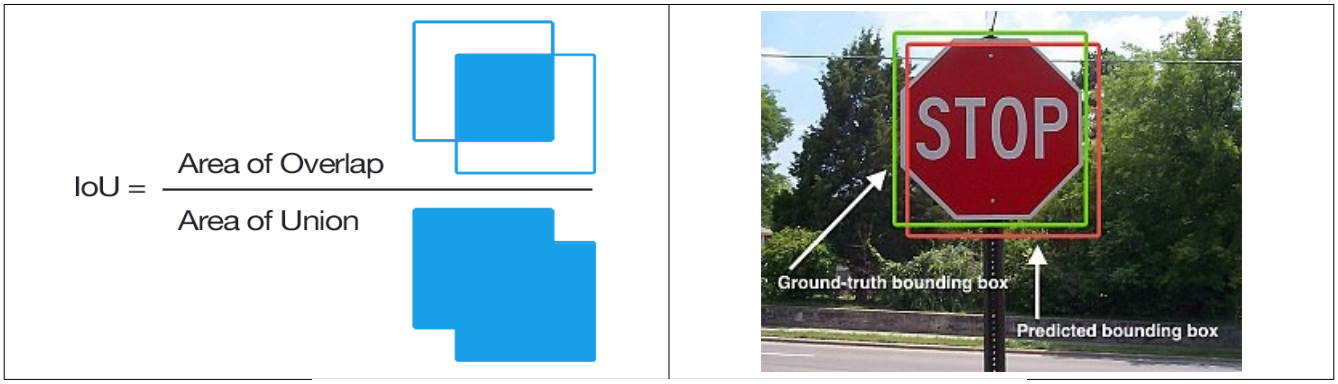


Figura: Intersecção sobre União na detecção de objetos.

Para cada objeto da imagem, existe uma caixa verdadeira geralmente desenhada por um ser humano e desconhecida pelo algoritmo. Considera-se que o algoritmo detectou corretamente esse objeto se ele propuser uma caixa cuja intersecção sobre união (IoU) das duas caixas for maior que um certo limiar l.

Usando a nomenclatura adotada, AX e QX são as imagens contendo objetos, e AY, QY e QP são conjuntos de caixas delimitadoras. O aprendiz vai receber uma imagem QX e vai gerar um conjunto de caixas QP baseando-se nos exemplos de treino AX e AY. Para calcular a qualidade do algoritmo, devemos comparar o conjunto de caixas verdadeiras QY e o conjunto de caixas QP geradas pelo algoritmo usando um certo limiar l para IoU. É possível contar as quantidades de casos TP, FP e FN (true positive, false positive e false negative):

- Um caso TP é quando IoU entre uma caixa verdadeira e uma caixa predita está acima do limiar.
- Um caso FP é quando não há nenhuma caixa verdadeira com IoU acima do limiar para uma caixa predita.

- Um caso FN é quando não há nenhuma caixa proposta com IoU acima do limiar para uma caixa verdadeira.

Na detecção de objetos, não há como contar casos TN. Assim, uma métrica que avalia a qualidade da detecção de objetos deve levar em consideração somente TP, FP e FN (e não considerar TN). Por exemplo, não é possível calcular taxa de acerto, taxa de erro, especificidade e ROC-AUC de um detector de objetos uma vez que, para calculá-las, é necessário conhecer TN. Também não é possível plotar uma curva ROC.

2.2 Precision Recall

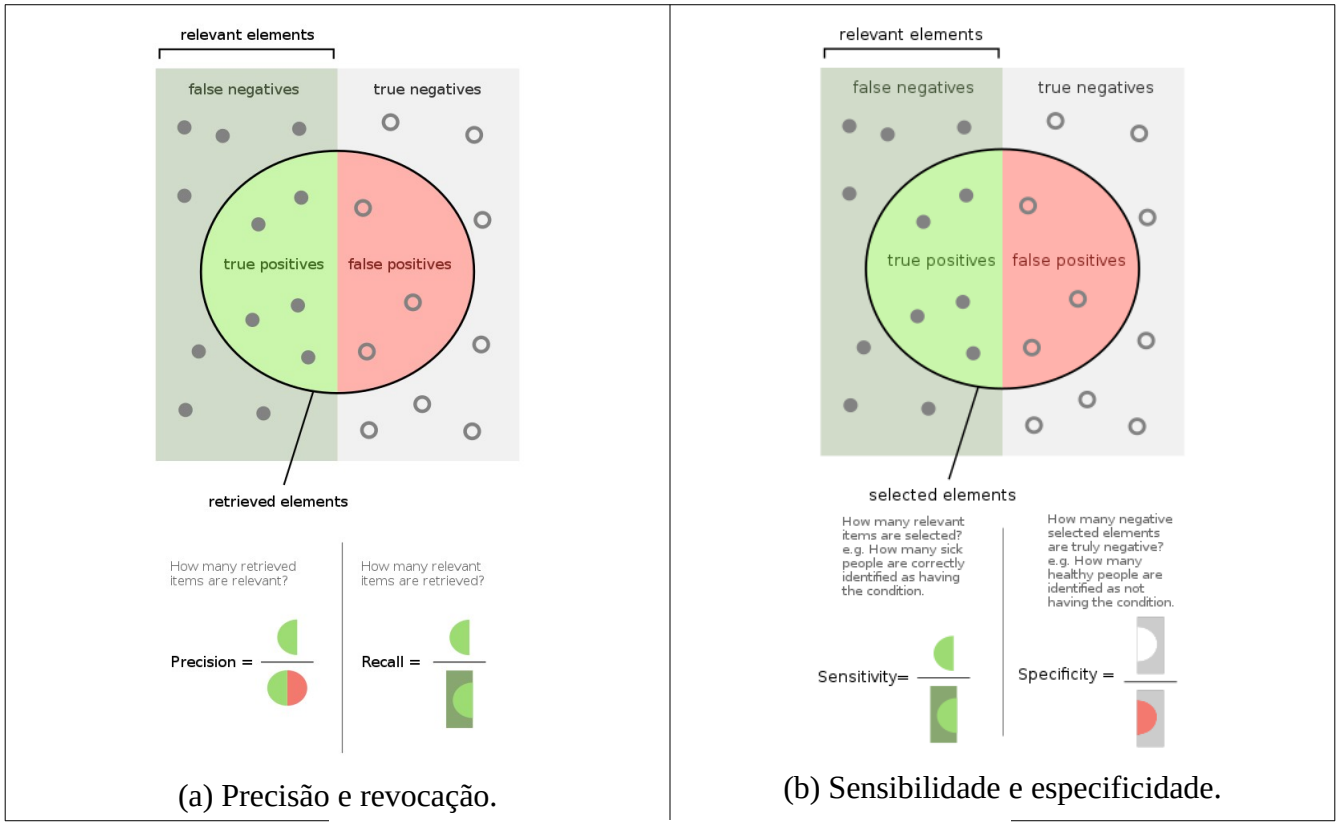
https://en.wikipedia.org/wiki/Sensitivity_and_specificity
https://en.wikipedia.org/wiki/Precision_and_recall

Porém, é possível calcular precisão e revocação de um detector de objetos. Definições (veja a figura F): P=TP+FN e N=FP+TN.

Precisão $\text{Precision} = \frac{TP}{TP+FP}$

Revocação ou sensibilidade $\text{Recall} = \text{Sensitivity} = \frac{TP}{TP+FN} = \frac{TP}{P}$

F1-score $\text{F1-Score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$



(a) Precisão e revocação.

(b) Sensibilidade e especificidade.

Figura F: Precision, recall, sensitivity and specificity.

Para que estes conceitos fiquem mais claros, considere que um teste para detectar uma certa doença foi avaliada em 2030 indivíduos com e sem doença, obtendo os seguintes resultados:

População total 2030	Teste positivo 200	Teste negativo 1830
Pacientes com doença P=30	TP=20	FN=10
Pacientes sem doença N=2000	FP=180	TN=1820

Dado que o teste deu positivo para um certo indivíduo, a precisão (10%) indica a probabilidade dele ter a doença.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{20}{200} = 10\,\%$$

Dado que um indivíduo tem a doença, a revocação ou sensibilidade (66,7%) indica a probabilidade do teste dar positivo.

$$\text{Recall} = \text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}} = \frac{20}{30} \approx 66.7\,\%$$

F1-Score é a média harmônica entre a precisão e a revocação:

$$\text{F1-Score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \times 0.1 \times 0.667}{0.1 + 0.667} = 0{,}173924381$$

2.3 Curva precision-recall (PRC) e precisão média (AP ou PR-AUC)

https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b#:~:text=Average%20precision%20is%20the%20area,is%20between%200%20to%201.

A curva precisão-revocação (PRC) é obtida variando limiar para todos os possíveis valores e plotando os pontos (recall, precision) resultantes (figura F2).

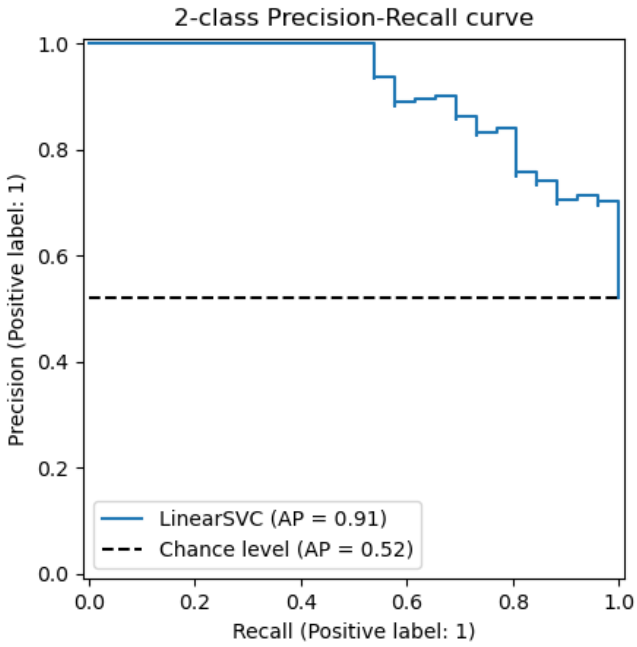


Figura F2: Exemplo de uma curva precision-recall (PRC).
https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#:~:text=The%20precision%2Drecall%20curve%20shows,a%20low%20false%20negative%20rate.

A área debaixo da curva precision-recall é chamada de PR-AUC (precision recall area under curve) ou AP (average precision, precisão média). É a precisão média quando se varia a revocação.

Na detecção de objetos, há uma AP diferente para cada classe de objeto. AP é uma métrica para avaliar a detecção de uma única classe de objeto. AP não calcula a média das precisões para diferentes classes de objetos.

2.4 Mean Average Precision (mAP)

https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b#:~:text=Average%20precision%20is%20the%20area,is%20between%200%20to%201

Mean average precision (mAP) é calculada tirando a média das AP de cada uma das classes de objetos.

$$\text{mAP} = \frac{1}{K} \sum_{i=1}^{K} \text{AP}_i$$

onde K é a quantidade de classes de objetos.

Esta é a métrica que costuma ser utilizada para avaliar a qualidade de detecção de objetos.

[PSI3472 aula 11, pular até aqui]

3. Convertendo classificador de imagens em detector de objetos

https://pyimagesearch.com/2020/06/22/turning-any-cnn-image-classifier-into-an-object-detector-with-keras-tensorflow-and-opencv/

Já estudamos (apostila classif-ead) o algoritmo de Viola-Jones que usa aprendizado de máquina clássico para detectar faces. Este algoritmo converte um modelo de classificação, que verifica se uma janela 24×24 contém uma face ou não, num detector de faces varrendo todas as posições e escalas da imagem. Eles usaram imagem integral para acelerar o cálculo dos filtros de Haar nas diferentes posições e escalas.

De forma semelhante, qualquer classificador de imagens (que classifica o conteúdo da janela como várias classes de objetos ou fundo) pode ser convertido em um detector de objetos.

3.1 Janela móvel.

Uma janela móvel percorre a imagem, localizando as regiões onde ocorrem os objetos buscados.

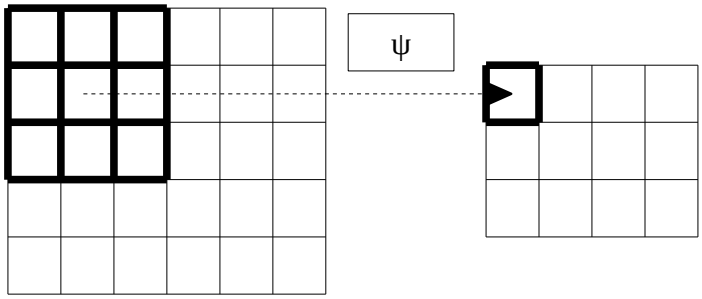


Figura 2: Janela móvel.

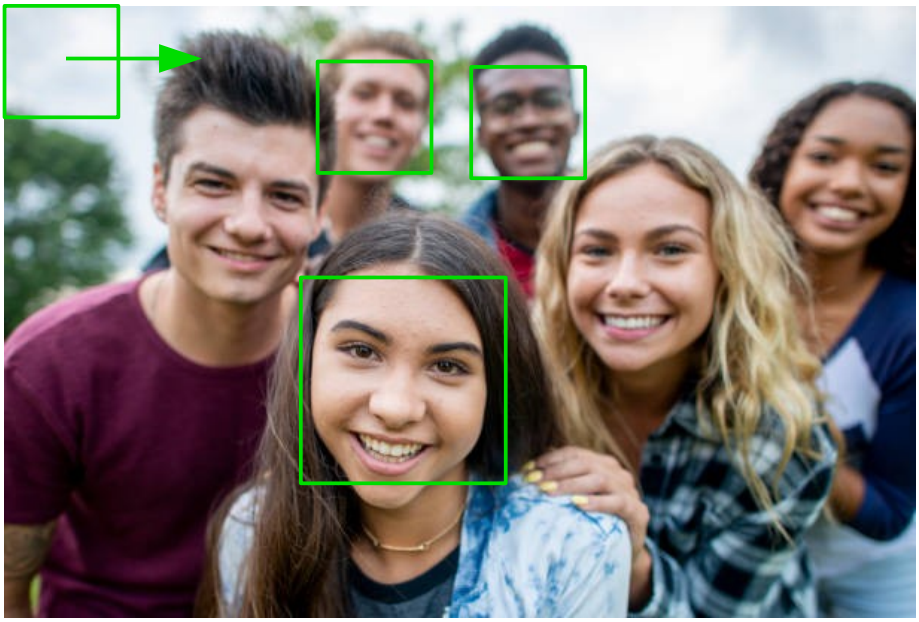


Figura 3: Usando janela móvel e classificando o conteúdo da janela móvel em objeto/background, é possível converter um classificador de imagem em detector de objetos.

3.2 Representação multi-escala.

Os objetos podem aparecer em diferentes escalas. Na figura 3, há rostos pequenos e grandes. Para detectar objetos em escalas diferentes, podemos usar duas estratégias:

a) Criar uma representação multi-escala da imagem. Isto é, redimensionar a imagem para várias escalas. O mais comum é criar a estrutura de pirâmide (figura 4) com imagem reduzida em fatores de 2. Detectar um objeto num nível k equivale a detectar o mesmo objeto 2^k vezes maior no nível zero.

Nota: Antes de reduzir a imagem por 2, costuma-se borrar a imagem com um filtro gaussiano para evitar aliasing (apostila transfgeo-ead).

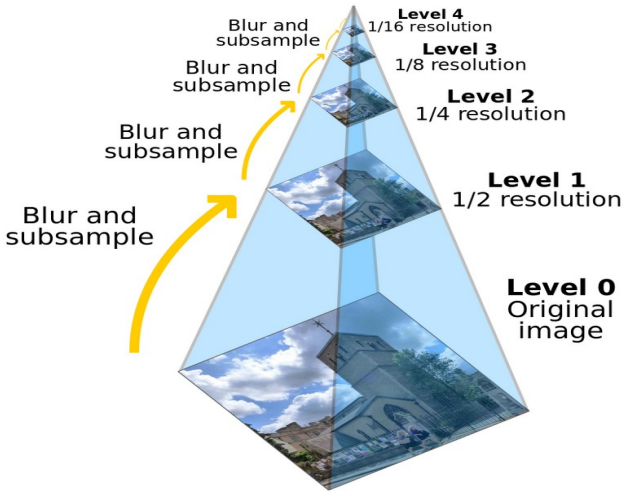


Figura 4: Representação multi-escala em pirâmida. Detectar um objeto no nível k+1 equivale a detectar objeto duas vezes maior no nível k.

b) Alterar o classificador de imagens para que detecte objetos de diferentes tamanhos. Por exemplo, é possível redimensionar o modelo em diferentes escalas em *template matching* para classificar imagens.

c) É possível combinar as estratégias (a) e (b) acima.

3.3 Supressão não-maximal.

É normal que um mesmo objeto seja detectado múltiplas vezes em diferentes posições e escalas (figura 5). Para evitar isto, é necessário executar alguma forma de supressão não-maximal. Supressão não-maximal escolhe, de um agrupamento de detecções, apenas a detecção com a maior confidência.

Figura 5: É necessário suprimir detecções não-maximais para evitar múltiplas detecções do mesmo objeto.

3.4 Detecção de objetos a partir de classificação de imagens

Juntando as ideias acima, é possível converter qualquer classificador de imagens (que classifica uma imagem em K+1 classes, com K tipos de objetos mais o background) num detector de objetos. Porém, esta ideia fica extremamente lenta usando CNN pois, para detectar objetos numa imagem, precisaria fazer centenas de milhares de inferências de CNN.

4. Proposta de região (region proposal)

https://pyimagesearch.com/2020/06/29/opencv-selective-search-for-object-detection/

O algoritmo proposto na seção anterior é *muito* lento, pois precisa classificar com CNN todas as regiões da imagem em todas as escalas.

Nota: Os algoritmos clássicos de detecção de objetos como *template matching* e Viola-Jones aceleram esta busca usando técnicas como FFT, filtros de Haar e imagem integral. Porém, essas técnicas não podem ser usadas para acelerar a predição de CNN.

Um algoritmo de proposta de regiões pode substitui a janela móvel e a representação multi-escala por um conjunto de regiões onde há possibilidade de aparecer os objetos de interesse (figuras F1 e 6).

Nota: Não tem problema se o algoritmo propuser algumas regiões que não contém nenhum objeto ou se propuser duas ou mais regiões para um mesmo objeto, pois esses falsos positivos serão filtrados posteriormente por CNN e pela supressão não-maximal. Porém, cada objeto de interesse deve estar obrigatoriamente contido em pelo menos uma região proposta. Se algum objeto não for proposto em nenhuma região, esse objeto não será detectado.

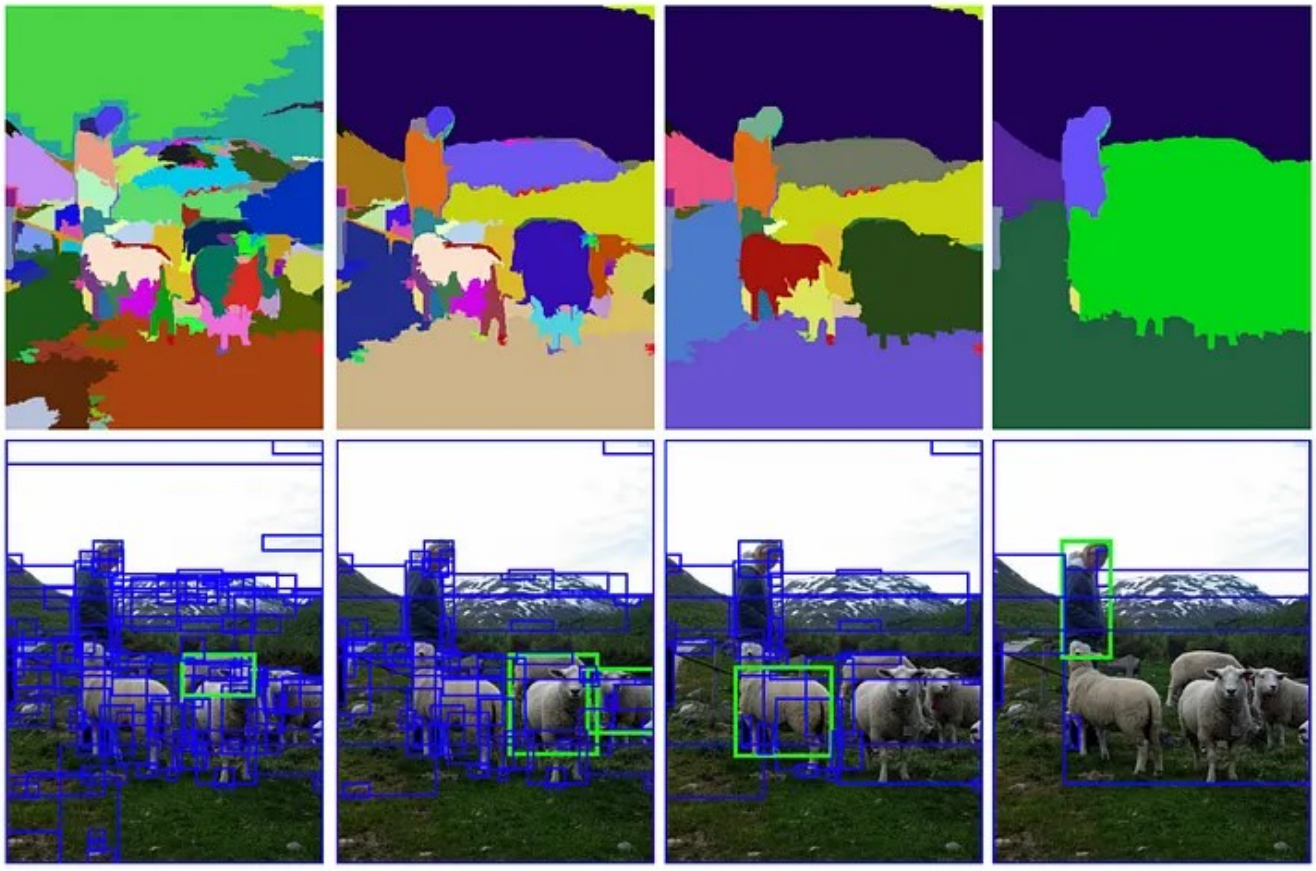


Figura F1: Exemplo de proposta de regiões [J.R.R. Uijlings and al. (2012)].

Para propor as regiões, o algoritmo agrupa as regiões com características semelhantes de cor, textura e outros atributos visuais (figura F1). As regiões extraídas pelo algoritmo alimentam uma CNN, que as classifica em objeto ou fundo. Esta ideia foi sendo melhorada ao longo do tempo nos algoritmos: (a) R-CNN, (b) fast R-CNN e (c) faster R-CNN:

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a

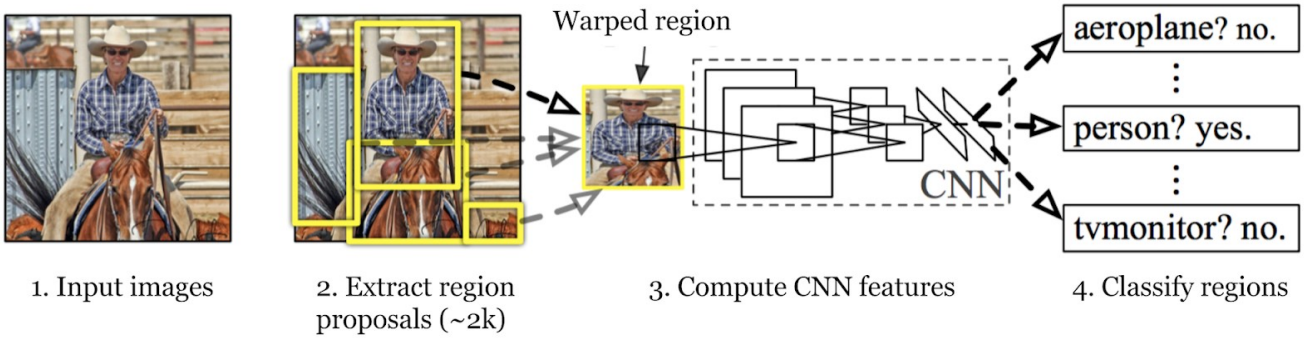


Figura 6: Proposta de região extrai regiões da imagem que podem ser objetos ou fundo.

5. R-CNN

Girshick apresenta R-CNN (Region-Based Convolutional Neural Network) para detectar objetos [Girshick2014]. R-CNN utiliza uma um algoritmo de proposta de regiões [Uijlings2013] para propor um conjunto de regiões de interesse (Regions of Interest - RoIs) que podem conter objetos. O algoritmo de proposta de regiões extrai RoIs da imagem que podem conter um objeto combinando pixels com cores e texturas semelhantes em caixas retangulares (figuras F1, 6 e 7).

No artigo, o algoritmo de proposta de regiões extrai 2.000 RoIs de cada imagem. Essas 2.000 RoIs são redimensionadas para um tamanho padrão e são repassadas para um modelo CNN pré-treinado que extrai 4096 atributos. Para cada classe de objetos *C*, uma SVM (Support Vector Machine) treinada para reconhecer especificamente objetos da classe *C* calcula a chance do vetor de atributos representar *C*. Em seguida, faz supressão não-maximal (figura 5).

Nota: Quase certamente, hoje em dia, a extração de atributos e a classificação seriam feitos de uma só vez usando somente CNN (sem usar SVM).

Além disso, R-CNN faz uma regressão, usando os 4096 atributos, para corrigir os erros de localização dos objetos cometidos pelo algoritmo de proposta de regiões.

Nota: Isto é feito de forma semelhante ao EP de PSI3472, 2024, que localiza bananas.

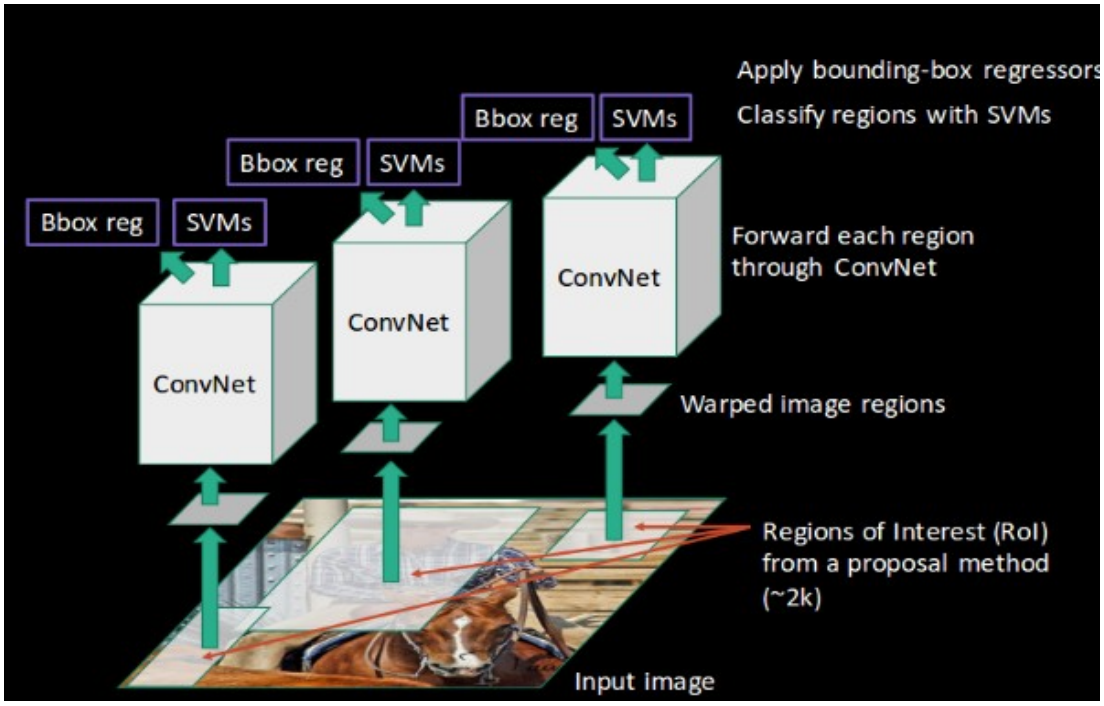


Figura 7: R-CNN usa proposta de regiões para obter regiões candidatos a conterem objetos. Essas regiões são classificadas usando CNNs em objeto ou fundo, juntamente com regressão de localização do objeto (figura de [Sharma2023]).

6. Fast R-CNN

Girshick apresenta, em [Girshick2015], Fast R-CNN que aumenta a velocidade de R-CNN.

Como vimos, R-CNN extrai os atributos das 2000 RoIs aplicando CNN a cada RoI redimensionado, totalizando 2000 inferências de CNN (figura 7). Fazer 2000 inferências de CNN é muito lento.

Com o conhecimento que temos sobre redes completamente convolucionais, fica evidente que não é necessário aplicar CNN 2000 vezes, uma vez para cada RoI: é possível extrair simultaneamente os atributos de todas as 2000 RoIs de uma única vez, usando uma rede completamente convolucional (figura 8). Apesar disso, Fast R-CNN continua dependendo de um algoritmo de proposta de regiões: deve receber na entrada a imagem juntamente com uma lista de 2000 RoIs.

A camada RoI-pooling é usada para converter os atributos de cada uma das RoIs para um tamanho padrão predefinido. Esta camada faz uma espécie de max-pooling adaptativo. Para facilitar a explicação, vamos supor que se definiu que a resolução espacial padrão dos atributos da RoI seja 7×7. Digamos que os atributos de uma RoI tenham resolução espacial 35×35. Neste caso, efetua-se um max-pooling 5×5 para diminuir a resolução desta RoI para o tamanho padrão 7×7.

Os atributos extraídos de cada RoI por RoI-pooling são repassados para uma rede neural densa que gera dois tipos de saídas: a classificação em k+1 categorias (k classes de objetos e o fundo) e a regressão para corrigir a localização do objeto, específica para cada classe. Este processo é efetuado sequencialmente, uma inferência da rede totalmente conectada para cada RoI.

O treino dessa rede com mini-batches de imagens é feito de forma hierárquica: primeiro amostra as imagens e depois amostra algumas ROIs dessas imagens. O treino procura minimizar simultaneamente os erros das tarefas de classificação e localização atribuindo pesos às duas tarefas.

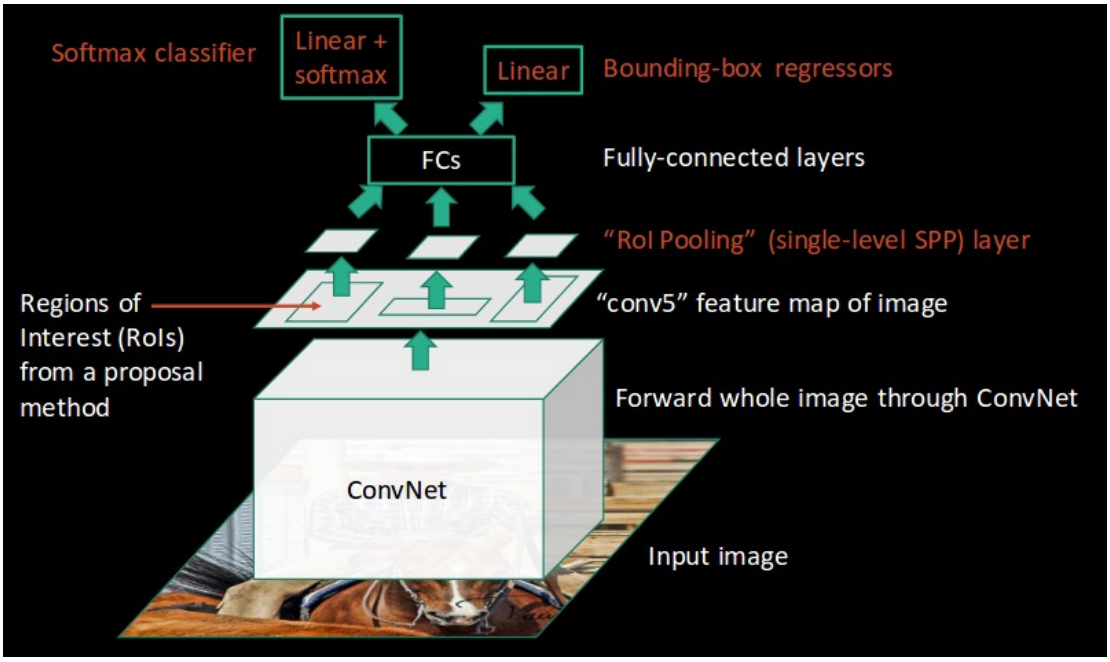


Figura 8: Fast R-CNN também usa proposta de regiões para obter RoIs candidatos a conterem objetos. Porém, a imagem é processada uma única vez com uma CNN que extrai os atributos de todas as RoIs simultaneamente. Os mapas de atributos correspondentes a diferentes RoIs são classificadas por um outro classificador (figura de [Sharma2023]).

7. Faster R-CNN

https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a
https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras
https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/
https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/

O artigo [Ren2016] (com R. Girshick como um dos co-autores) introduziu Faster R-CNN, que eliminou a necessidade do algoritmo de proposta de regiões (usado por R-CNN e Fast R-CNN), que era o gargalo de velocidade. No seu lugar, usa uma rede convolucional denominada de RPN (Region Proposal Network, figura 9) para propor as regiões candidatas. Porém, continua fazendo a inferência em 2 passos:

a) Proposta de regiões de interesse (RoIs) pela RPN.
b) Classificação das RoIs e a localização precisa dos objetos, usando o mesmo algoritmo de Fast R-CNN.

RPN e Fast R-CNN compartilham os mesmos “conv layers” (figura 9) que geram o mapa de atributos (isto acelera tanto o treino como a predição). Diferentes modelos de CNN podem ser usados como “conv layers”. Nesta apostila, vamos adotar VGG16, um dos modelos usados no artigo original. VGG16, sem as camadas de topo, reduz as dimensões espaciais da imagem de entrada por 16 e extrai 512 atributos para cada elemento espacial do mapa de atributos. Isto é, se a imagem de entrada tem dimensão 288×288×3 então a saída será 18×18×512 (sem as camadas de topo). Vamos assumir essas dimensões da imagem para facilitar a explicação.

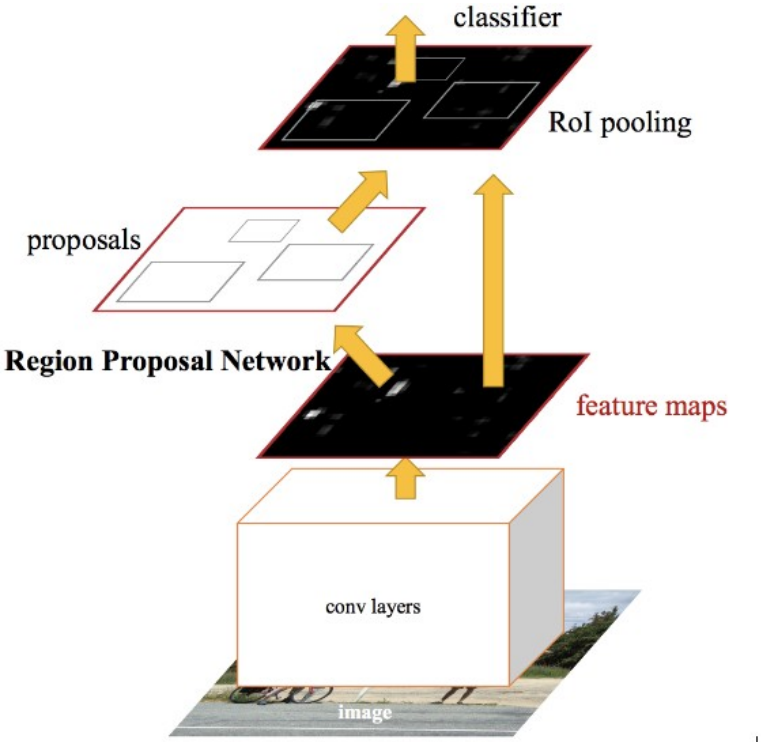


Figura 9: No Faster R-CNN, uma rede CNN chamada Region Proposal Network (RPN) propõe as regiões de interesse (figura de [Sharma2023]).

Faster R-CNN introduz o conceito de “caixas de âncora” para que RPN possa propor RoIs com diferentes resoluções e formatos. Para cada posição espacial da janela móvel no mapa de atributos (o artigo original usa 3×3, figura 10), associam-se *k* âncoras (o artigo usa *k*=9 âncoras com 3 escalas e 3 formatos diferentes - figura 11).

Podemos dizer que as âncoras “não possuem existência real” dentro de RPN – elas são usadas somente para verificar se contêm ou não objetos de interesse calculando a intersecção com caixa delimitadora verdadeira. O uso de âncoras elimina a necessidade de processar a imagem em diferentes escalas. Os autores observam que o campo receptivo na imagem de entrada correspondente à janela 3×3 no mapa de atributos é grande: 228×228 pixels usando VGG16. Isto permite que a janela 3×3 no mapa de atributos “enxergue” objetos grandes na imagem de entrada.

Para levar em consideração janela 3×3 no mapa de atributos, RPN acrescenta mais uma camada convolucional 3×3 após a última camada de VGG16. Essa camada recebe um mapa de atributos 18×18×512 e gera um outro mapa 18×18×512 (isto é, sem alterar as dimensões). Depois, há mais (1+4)×9=45 convoluções 1×1 que irão gerar 45 saídas para cada um dos 18×18 elementos espaciais do mapa de atributos.

A primeira saída [(**1**+4)×9] é a saída da rede de classificação (*cls*) com ativação sigmoide predizendo se dentro da caixa de âncora há um objeto ou não. O artigo original usou 2 saídas com ativação softmax, porém como temos uma classificação binária (objeto/fundo) o mesmo pode ser feito usando uma única saída. As quatro saídas restantes [(1+**4**)×9] são da rede regressão (*reg*) que corrige a localização do objeto dentro da âncora. Como em cada posição espacial há 9 âncoras, há a multiplicação por 9 [(1+4)×**9**].

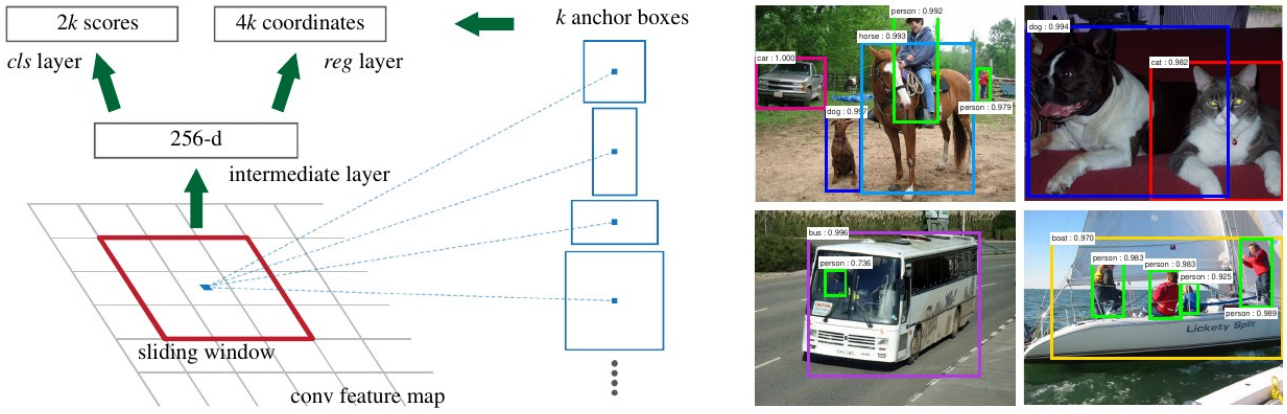


Figura 10: Para cada posição da janela móvel, há *k* âncoras associadas (*k*=9 no artigo). Uma âncora é uma região de interesse em diferentes escalas e formatos (figura de [Ren2016]).

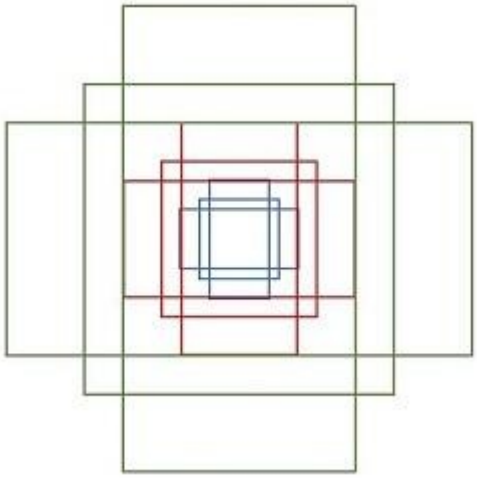


Figura 11: 9 âncoras de Faster R-CNN com 3 escalas e 3 formatos (figura de [https://www.cnblogs.com/dmyu/p/9738855.html])

RPN é treinada de ponta a ponta usando back-propagation e descida do gradiente. Para treinar RPN, é necessário calcular, para cada caixa delimitadora verdadeira, se esta possui uma intersecção significativa com cada uma das 9 âncoras em cada uma das 18×18 posições. As âncoras com (sem) intersecção significativa são consideradas âncoras positivas (negativas).

Cada mini-batch é formada por uma única imagem de treino com âncoras positivas e negativas. Os autores dizem que poderia ter otimizado a função de perda usando todas as âncoras, mas isto enviesaria o treino pois as imagens normalmente possuem muito mais âncoras negativas do que positivas. Assim, os autores escolhem aleatoriamente 128 âncoras positivas e 128 negativas para formar um mini-batch (estes números podem mudar se não houver 128 âncoras positivas na imagem). Este mapa de saída é usado como saída desejada para treinar RPN.

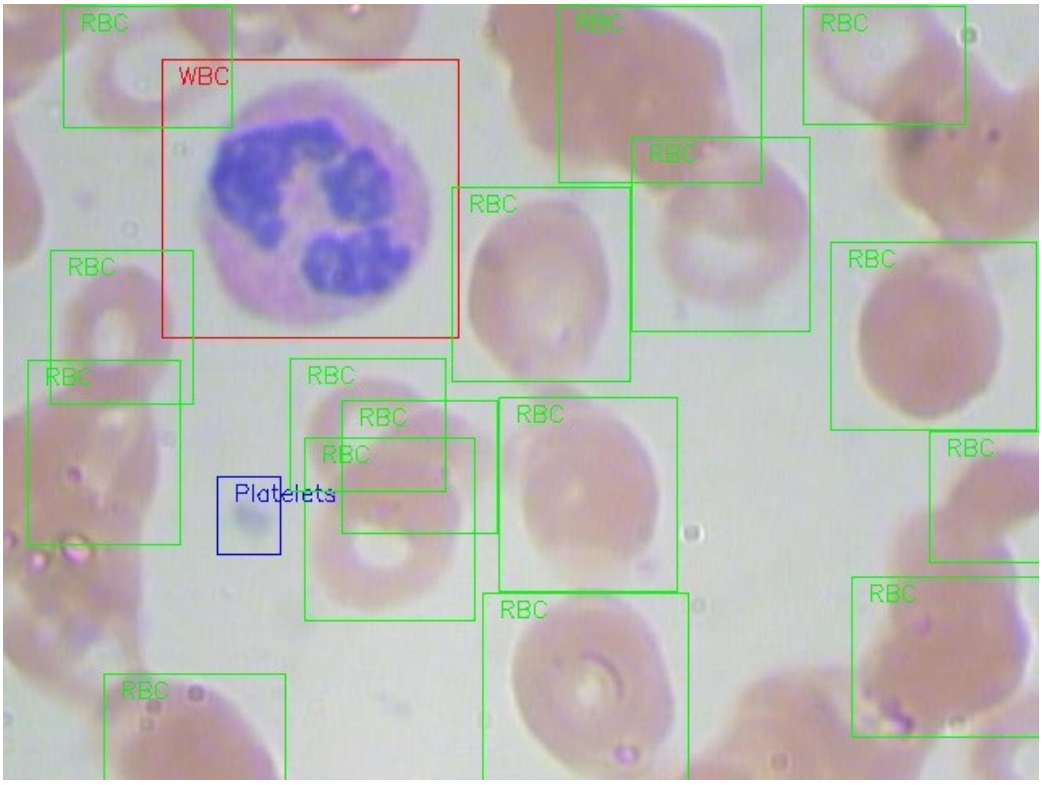
As camadas “conv layers” (figura 9) são compartilhadas pelas redes RPN e Fast R-CNN. Os autores alternam os treinos entre as duas redes. Primeiro, RPN é treinada para propor RoIs corretas. As camadas “conv layers” assim obtidas são ajustadas (fine tuning) pelo Fast R-CNN para detectar e localizar os objetos de interesse. E este processo é iterado.

8. Uma implementação de Faster R-CNN

Xu apresenta uma implementação de Faster R-CNN escrito em Keras e o treina para detectar pessoas, celulares e carros:
Blog: https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a
Implementação: https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras

Nota: A implementação tinha problemas de compatibilidade com as bibliotecas atuais de Keras (2.15.0), Pandas, etc. Fiz correções para que funcionasse. O programa privado modificado por mim está em:
https://drive.google.com/drive/folders/1I228NhUwz08j3KBtZvA4rdzq0wsfzVzk
Não treinei durante muitas épocas e o modelo comete muitos erros.

Para não perder muito tempo, treinei o modelo usando um conjunto de imagens mais simples, para detectar células sanguíneas: glóbulos vermelhos, glóbulos brancos e plaquetas:
https://github.com/Shenggan/BCCD_Dataset



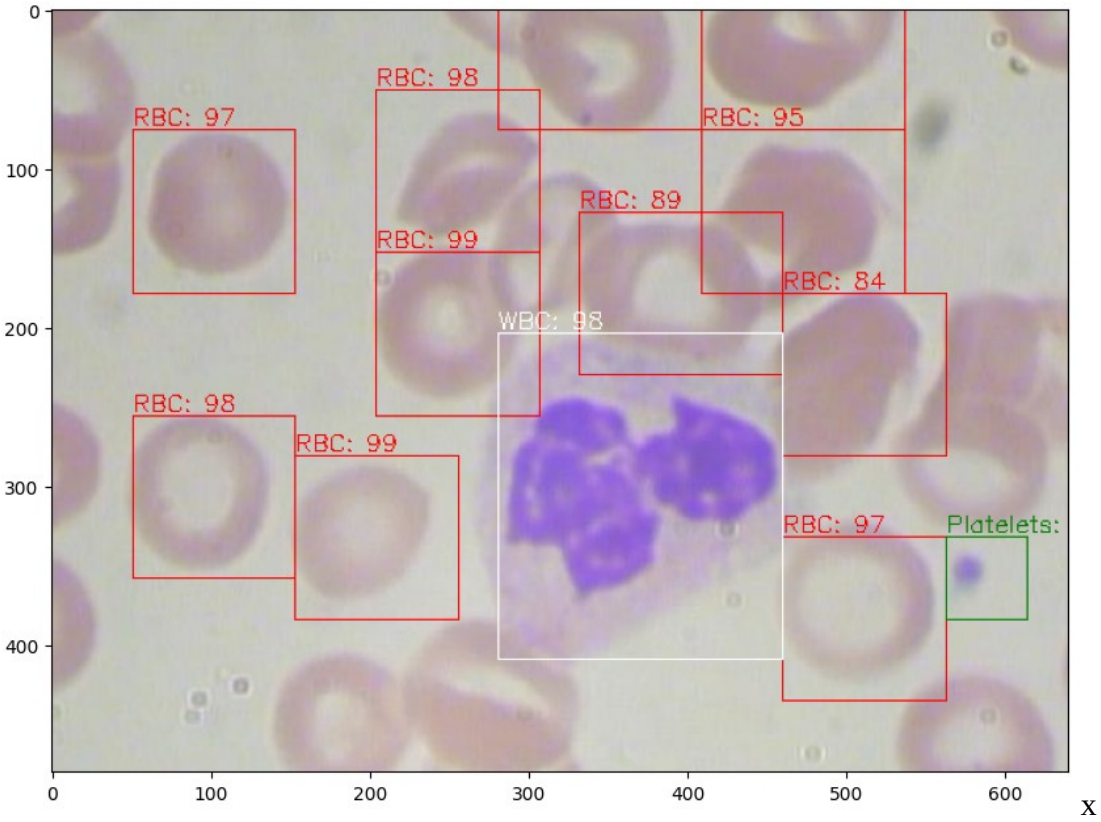


Figura F3: Exemplos de imagem do conjunto BCCD_Dataset.

A solução está em (inacessível para público):

https://drive.google.com/drive/folders/1l9S7J72O-xbYMyK-jsZikJ9CXM6kr4AO (pasta)
https://drive.google.com/file/d/18V1utCFjme3Y03b-HC1fBjHNuna4BvgR/view?usp=drive_link (treino)
https://drive.google.com/file/d/1xdpyS0viQPHqDYNQnxrWc9Vm6WbHM-HZ/view?usp=drive_link (teste)

Observações sobre a implementação:

1) O programa *export.py* lê os arquivos *XML* do diretório *BCCD/Annotations* e converte em *bbox.csv*.

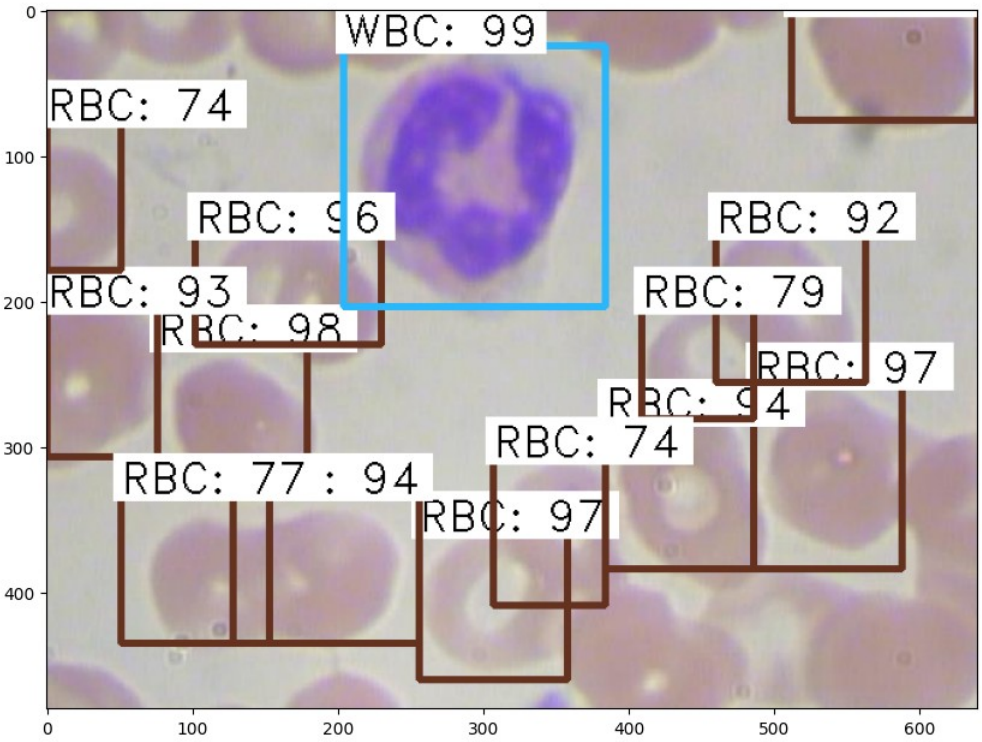
2) O meu programa organiza *bbox.csv* em arquivos *annotation.txt* (treino) e *test_annotation.txt*, no formato que o programa *frcnn* entende.

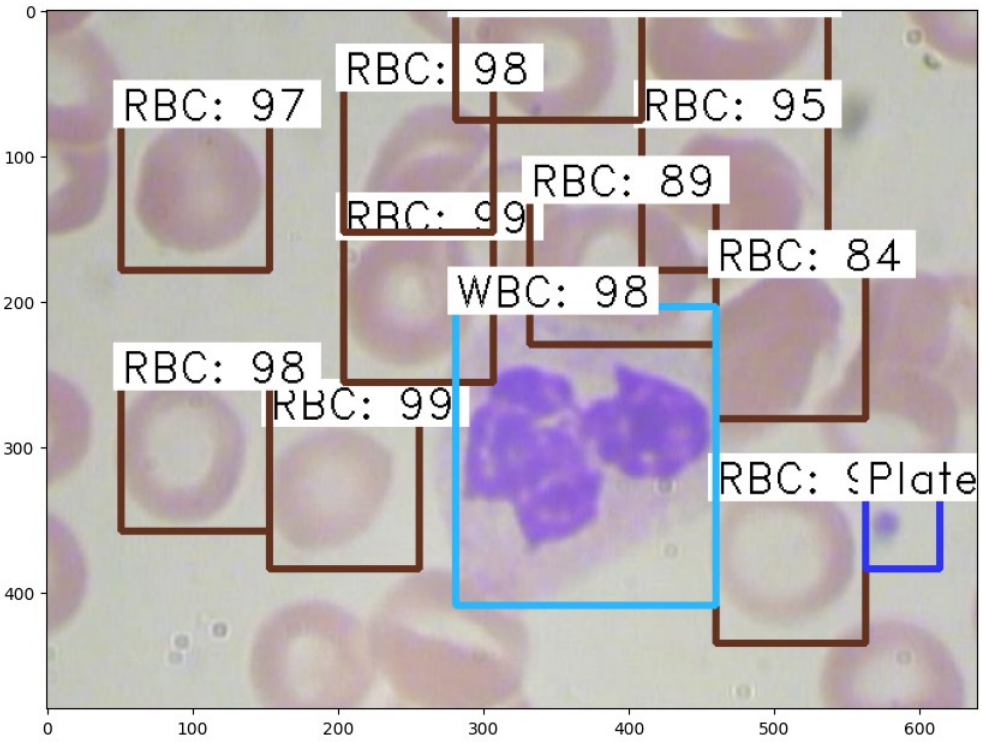
3) O programa *frcnn_train_vgg.ipynb* lê os arquivos:
   a) annotation.txt
   b) model/vgg16_weights_tf_dim_ordering_tf_kernels.h5 (peso inicial vgg16, se não existir, treina do zero). Veja https://notebook.community/jessicaowensby/We-Rise-Keras/notebooks/03_VGG16 para ver como baixar e usar esses pesos.
   c) model/model_frcnn_vgg.hdf5 (se está treinando de onde parou).

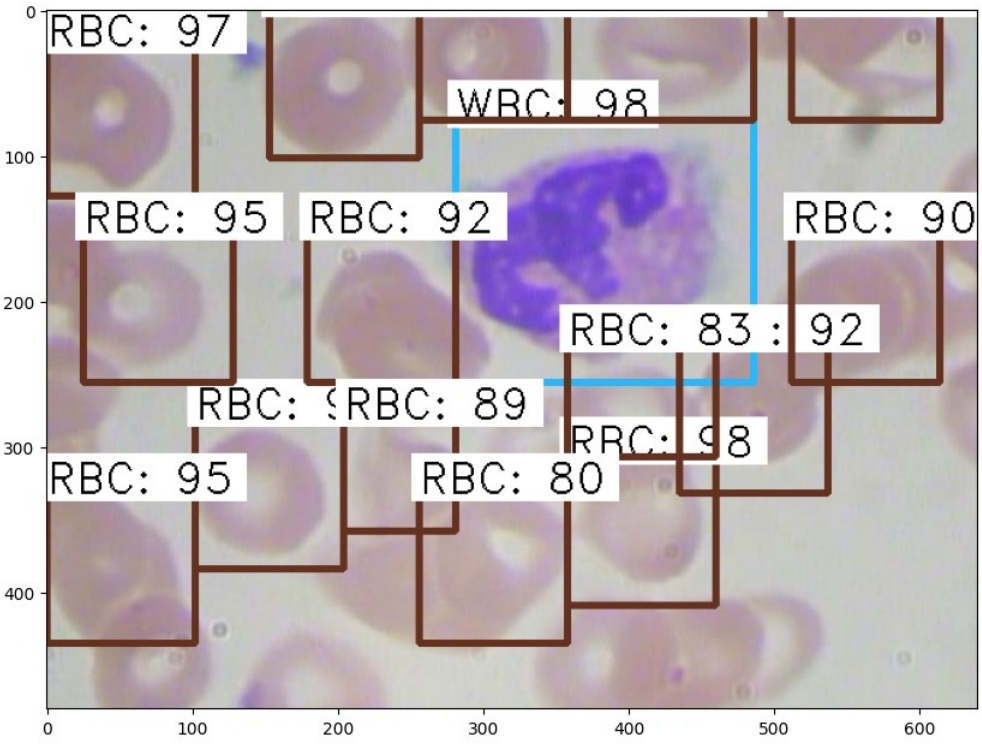
E gera:
   a) model_vgg_config.pickle
   b) model/model_frcnn_vgg.hdf5
   c) model/record.csv (salva as perdas por época)

4) O programa *frcnn_test_vgg.ipynb* lê os arquivos:
   a) test_annotation.txt
   b) model_vgg_config.pickle
   c) model/model_frcnn_vgg.hdf5
   d) model/record.csv

Algumas saídas:







A mean average precision obtida foi 0.745.

[PSI3472 aula 11, fim]

[PSI3472 aula 12, início]

R-CNN e YOLO

[https://machinelearningmastery.com/object-recognition-with-deep-learning/]

Basicamente, existem duas famílias de métodos CNN para detecção de objetos:

- R-CNN: Consiste em propor regiões candidatas e depois verificar se há algum objeto nessas regiões.
- YOLO: Consiste de uma única rede que recebe uma imagem de entrada e prediz diretamente retângulos com rótulos.

Os modelos R-CNN são geralmente mais precisos, mas a família de modelos YOLO costuma ser muito mais rápida que o R-CNN, alcançando a detecção de objetos em tempo real.

Referências

Demonstração de Yolo que segmenta carros:
https://medium.com/@hichengkang/auto-labeling-is-not-a-dream-a250af040102

No meu computador, esse programa está em ~/haepi/deep/keras/yolo/v8/cars.py
Outro exemplo: ~/haepi/deep/keras/yolo/v8/stop.py

Site de Ultralytics com modelos pré-treinados:
https://docs.ultralytics.com/

Detecção em Keras
https://pyimagesearch.com/2020/07/13/r-cnn-object-detection-with-keras-tensorflow-and-deep-learning/

Explicação simplificada de R-CNN, fast R-CNN e faster R-CNN:
https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a

Introdução a object detection
https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning

Explicação de faster R-CNN:
https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection

Introdução à detecção de objetos usando KerasCV:
https://keras.io/guides/keras_cv/object_detection_keras_cv/

Detecção de objetos com RetinaNet em Keras:
https://keras.io/examples/vision/retinanet/

YoloV8:
https://keras.io/examples/vision/yolov8/

[PSI3472 aula 12, fim]