Classificação e localização de objeto:

Gato

Digamos que temos rede neural que reconhece cão, gato e pássaro: Rede tem 3 saídas - cão, gato e pássaro.

1) Treinar rede com mais 4 saídas: xcentro, ycentro, largura, altura.

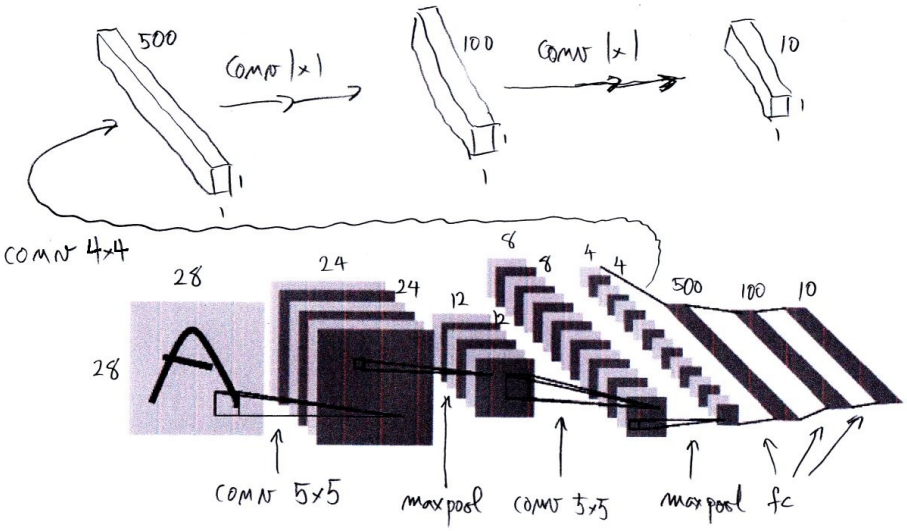
2) Janela móvel. Tem que treinar com 4 categorias: cão, gato, pássaro e "não cão, gato ou pássaro". Lento.

Tem jeito melhor?

3) R-CNN.
Region proposal: Propõe regiões onde podem ter objetos.
Classifier: Verifica se nessa região tem algum objeto.

Rede completamente convolucional
Como modificar rede neural convolucional para localizar um objeto sem rodar janela móvel?

1) Rodar rede neural convolucional em janela móvel. Problema: É lento.
2) Dá para fazer a mesma coisa que rodar rede neural em janelas, eliminando as camadas densas (ficam só camadas convolucionais).



LeCun et al. 1998

LeNet original que aceita imagem 28x28:
1x28x28 → conv5x5 → 20x24x24 → maxpool → 20x12x12 → conv5x5 → 40x8x8 → maxpool → 40x4x4 → flatten → dense → 500 → dense → 100 → dense → 10
A saída possui 10 elementos, referentes a 10 categorias. Não se pode entrar imagens maiores que 28x28 nesta rede.

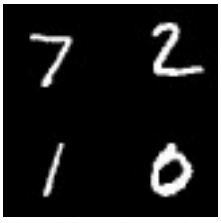
Trocando camadas densas (fc) por convolucionais:
1x28x28 → conv5x5 → 20x24x24 → maxpool → 20x12x12 → conv5x5 → 40x8x8 → maxpool → 40x4x4 → conv4x4 → 500x1x1 → conv1x1 → 100x1x1 → conv1x1 → 10x1x1
Esta camada faz exatamente a mesma coisa que LeNet original. A saída possui 10 imagens 1x1 referentes a 10 categorias.
(qx0.png) 

```
[[7]]
```

O que acontece se entrar uma imagem maior que 28x28? Por exemplo, 32x32?
1x32x32 → conv5x5 → 20x28x28 → maxpool → 20x14x14 → conv5x5 → 40x10x10 → maxpool → 40x5x5 → conv4x4 → 500x2x2 → conv1x1 → 100x2x2 → conv1x1 → 10x2x2
A saída será um tensor 10x4x4. Isto equivale a aplicar LeNet original 4 vezes, fazendo stride de 4.
(qx0b.png) 

```
[[ 7  7]
 [ 7 -1]]
```

O que acontece se entrar uma imagem 80x80?
1x80x80 → conv5x5 → 20x76x76 → maxpool → 20x38x38 → conv5x5 → 40x32x32 → max-pool → 40x16x16 → conv4x4 → 500x12x12 → conv1x1 → 100x12x12 → conv1x1 → 10x12x12

(q2.png) 

```
[[-1  7  7 -1 -1 -1 -1 -1 -1  5  7 -1  2  2]
 [ 7  7  7  7  7 -1 -1 -1 -1 -1  2  2  2 -1]
 [-1  7 -1 -1 -1 -1 -1 -1 -1  2 -1 -1 -1 -1]
 [-1  1  4 -1 -1 -1 -1 -1 -1 -1 -1  4  4  4]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1  5  5  5  7]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  7  7]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  2  2]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1  7 -1  2 -1]
 [ 7  1  1 -1 -1 -1 -1 -1 -1 -1 -1  6 -1 -1]
 [-1  1 -1 -1 -1 -1 -1 -1 -1  4  1  0  0 -1]
 [ 1  1 -1 -1 -1 -1 -1 -1 -1  4 -1 -1 -1  9]]
```

Teria que treinar com categoria "não-dígito".

```
#detect1.py - treino
import keras
from keras.datasets import mnist
from keras.models import Sequential, load_model
from keras.layers import Dropout, Conv2D, MaxPooling2D, Dense, Flatten
from keras import optimizers
import numpy as np
import cv2;

batch_size = 100
num_classes = 10
epochs = 30

# input image dimensions
nl, nc = 28, 28

# the data, split between train and test sets
(ax, ay), (qx, qy) = mnist.load_data()

#print('channels_last - formato de TensorFlow');
ax = ax.reshape(ax.shape[0], nl, nc, 1)
qx = qx.reshape(qx.shape[0], nl, nc, 1)
input_shape = (nl, nc, 1)

ax = ax.astype('float32')
qx = qx.astype('float32')
ax /= 255 #0 a 1
qx /= 255 #0 a 1
print(ax.shape[0], 'train samples')
print(qx.shape[0], 'test samples')

# convert class vectors to binary class matrices
ay = keras.utils.to_categorical(ay, num_classes)
ay = ay.reshape(ay.shape[0],1,1,num_classes);
qy = keras.utils.to_categorical(qy, num_classes)
qy = qy.reshape(qy.shape[0],1,1,num_classes);

#Execute se nao houver arquivo detect1.h5
model = Sequential(); #Entrada: 28*28*1
model.add(Conv2D(20, kernel_size=(5,5), activation='relu', input_shape=input_shape)); #Saida0: 20*24*24
model.add(MaxPooling2D(pool_size=(2,2))); #Saida1: 20*12*12
model.add(Conv2D(40, kernel_size=(5,5), activation='relu')); #Saida2: 40*8*8
model.add(MaxPooling2D(pool_size=(2,2))); #Saida3: 40*4*4
#
#LeNet original
#model.add(Flatten()); #Saida4: 640
#model.add(Dense(500, activation='relu')); #Saida5: 500
#model.add(Dense(100, activation='relu')); #Saida6: 100
#model.add(Dense(num_classes, activation='softmax')); #Saida7: 10
#
#Trocando camadas densas por convolucionais
model.add(Conv2D(500, kernel_size=(4,4), activation='relu')) #Saida5: 500*1*1
model.add(Conv2D(100, kernel_size=(1,1), activation='relu')) #Saida6: 100*1*1
model.add(Conv2D(num_classes, kernel_size=(1,1), activation='softmax')) #Saida7: 10*1*1
"""
#Execute se houver arquivo detect1.h5
model=load_model('detect1.h5')
"""

#from keras.utils import plot_model
#plot_model(model, to_file='detect1.png', show_shapes=True)
#from keras.utils import print_summary
#print_summary(model)

opt=optimizers.Adam(amsgrad=True)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(ax, ay,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(qx, qy))

score = model.evaluate(qx, qy, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

model.save('detect1.h5')
```

Epoch 30 - acc: 1.0000 - val_acc: 0.9936

```
#localize.py
import keras
from keras.datasets import mnist
from keras.models import Sequential, load_model;
from keras.layers import Dropout, Conv2D, MaxPooling2D, Dense, Flatten
from keras import optimizers
import numpy as np
import cv2;
import sys;

if len(sys.argv)!=2:
  print("localize.py ent.png");
  sys.exit()

num_classes = 10

qx0=cv2.imread(sys.argv[1],0);
nl, nc = qx0.shape[0], qx0.shape[1]
input_shape = (nl, nc, 1)

model = Sequential(); #Entrada: 28*28*1
model.add(Conv2D(20, kernel_size=(5,5), activation='relu', input_shape=input_shape)); #Saida0:
20*24*24
model.add(MaxPooling2D(pool_size=(2,2))); #Saida1: 20*12*12
model.add(Conv2D(40, kernel_size=(5,5), activation='relu')); #Saida2: 40*8*8
model.add(MaxPooling2D(pool_size=(2,2))); #Saida3: 40*4*4
model.add(Conv2D(500, kernel_size=(4,4), activation='relu')) #Saida5: 500*1*1
model.add(Conv2D(100, kernel_size=(1,1), activation='relu')) #Saida6: 100*1*1
model.add(Conv2D(num_classes, kernel_size=(1,1), activation='softmax')) #Saida7: 10*1*1
model.load_weights("detect1.h5");

qx0 = qx0.astype("float32");
qx0 /= 255 #0 a 1
qx0=qx0.reshape(1,nl,nc,1);
qp = model.predict(qx0);
print(qp.shape)

snl, snc = qp.shape[1], qp.shape[2];
qp = qp.reshape(snl,snc,10);
sai=np.empty([snl,snc],dtype=np.int8);
for l in range(snl):
  for c in range(snc):
    i=np.argmax(qp[l,c,:])
    if qp[l,c,i]>0.999:
      #print(l,c,i,qp[l,c,i]);
      sai[l,c]=i;
    else:
      sai[l,c]=-1;
print(sai);
#print(qp);
```

(PSI3472-2019 Aula 7 exercício 4) Modifique o programa caogato1.py (aula5, exercício 3, da apostila cifar-reduzido) para obter programa que aceita imagens coloridas maiores que 32x32. Teste esse programa na imagem caogato.png que possui dimensão 80x80.

caogato.png

YOLO object detection (You Only Look Once)

Site original do YOLO (escrito em C):
`https://pjreddie.com/darknet/yolo/`

Como converter para Keras:
`https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/`
`https://github.com/experiencor/keras-yolo3`

Pesos convertidos em:
`http://www.lps.usp.br/hae/apostila/keras-yolov3.zip`

```
# detect.py
# load yolov3 model and perform object detection
# based on https://github.com/experiencor/keras-yolo3
import numpy as np
from numpy import expand_dims
from keras.models import load_model
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from matplotlib import pyplot
from matplotlib.patches import Rectangle
import sys

class BoundBox:
    def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
        self.xmin = xmin
        self.ymin = ymin
        self.xmax = xmax
        self.ymax = ymax
        self.objness = objness
        self.classes = classes
        self.label = -1
        self.score = -1

    def get_label(self):
        if self.label == -1:
            self.label = np.argmax(self.classes)

        return self.label

    def get_score(self):
        if self.score == -1:
            self.score = self.classes[self.get_label()]

        return self.score

def _sigmoid(x):
    return 1. / (1. + np.exp(-x))

def decode_netout(netout, anchors, obj_thresh, net_h, net_w):
    grid_h, grid_w = netout.shape[:2]
    nb_box = 3
    netout = netout.reshape((grid_h, grid_w, nb_box, -1))
    nb_class = netout.shape[-1] - 5
    boxes = []
    netout[..., :2]  = _sigmoid(netout[..., :2])
    netout[..., 4:]  = _sigmoid(netout[..., 4:])
    netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]
    netout[..., 5:] *= netout[..., 5:] > obj_thresh

    for i in range(grid_h*grid_w):
        row = i / grid_w
        col = i % grid_w
        for b in range(nb_box):
            # 4th element is objectness score
            objectness = netout[int(row)][int(col)][b][4]
            if(objectness.all() <= obj_thresh): continue
            # first 4 elements are x, y, w, and h
            x, y, w, h = netout[int(row)][int(col)][b][:4]
            x = (col + x) / grid_w # center position, unit: image width
            y = (row + y) / grid_h # center position, unit: image height
            w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width
            h = anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height
            # last elements are class probabilities
            classes = netout[int(row)][col][b][5:]
            box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
            boxes.append(box)
    return boxes

def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
    new_w, new_h = net_w, net_h
    for i in range(len(boxes)):
        x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
        y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
```

```
            boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
            boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
            boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
            boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)

    def _interval_overlap(interval_a, interval_b):
        x1, x2 = interval_a
        x3, x4 = interval_b
        if x3 < x1:
            if x4 < x1:
                return 0
            else:
                return min(x2,x4) - x1
        else:
            if x2 < x3:
                 return 0
            else:
                return min(x2,x4) - x3

    def bbox_iou(box1, box2):
        intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
        intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
        intersect = intersect_w * intersect_h
        w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
        w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin
        union = w1*h1 + w2*h2 - intersect
        return float(intersect) / union

    def do_nms(boxes, nms_thresh):
        if len(boxes) > 0:
            nb_class = len(boxes[0].classes)
        else:
            return
        for c in range(nb_class):
            sorted_indices = np.argsort([-box.classes[c] for box in boxes])
            for i in range(len(sorted_indices)):
                index_i = sorted_indices[i]
                if boxes[index_i].classes[c] == 0: continue
                for j in range(i+1, len(sorted_indices)):
                    index_j = sorted_indices[j]
                    if bbox_iou(boxes[index_i], boxes[index_j]) >= nms_thresh:
                        boxes[index_j].classes[c] = 0

    # load and prepare an image
    def load_image_pixels(filename, shape):
        # load the image to get its shape
        image = load_img(filename)
        width, height = image.size
        # load the image with the required size
        image = load_img(filename, target_size=shape)
        # convert to numpy array
        image = img_to_array(image)
        # scale pixel values to [0, 1]
        image = image.astype('float32')
        image /= 255.0
        # add a dimension so that we have one sample
        image = expand_dims(image, 0)
        return image, width, height

    # get all of the results above a threshold
    def get_boxes(boxes, labels, thresh):
        v_boxes, v_labels, v_scores = list(), list(), list()
        # enumerate all boxes
        for box in boxes:
            # enumerate all possible labels
            for i in range(len(labels)):
                # check if the threshold for this label is high enough
                if box.classes[i] > thresh:
                    v_boxes.append(box)
                    v_labels.append(labels[i])
                    v_scores.append(box.classes[i]*100)
                    # don't break, many labels may trigger for one box
        return v_boxes, v_labels, v_scores

    # draw all results
    def draw_boxes(filename, v_boxes, v_labels, v_scores):
        # load the image
        data = pyplot.imread(filename)
        # plot the image
        pyplot.imshow(data)
        # get the context for drawing boxes
        ax = pyplot.gca()
        # plot each box
        for i in range(len(v_boxes)):
            box = v_boxes[i]
            # get coordinates
            y1, x1, y2, x2 = box.ymin, box.xmin, box.ymax, box.xmax
            # calculate width and height of the box
            width, height = x2 - x1, y2 - y1
            # create the shape
            rect = Rectangle((x1, y1), width, height, fill=False, color='white')
```

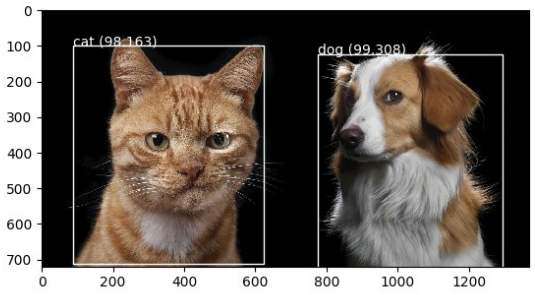
```
            # draw the box
            ax.add_patch(rect)
            # draw text and score in top left corner
            label = "%s (%.3f)" % (v_labels[i], v_scores[i])
            pyplot.text(x1, y1, label, color='white')
        # show the plot
        pyplot.show()

if len(sys.argv)!=2:
  print("detect nomeimg.ext");
  sys.exit();
# load yolov3 model
model = load_model('yolov3.h5')
# define the expected input shape for the model
input_w, input_h = 416, 416 #320 ou 416 ou 608
# define our new photo
photo_filename = sys.argv[1]
# load and prepare image
image, image_w, image_h = load_image_pixels(photo_filename, (input_w, input_h))
# make prediction
yhat = model.predict(image)
# summarize the shape of the list of arrays
#print([a.shape for a in yhat])
# define the anchors
anchors = [[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]
# define the probability threshold for detected objects
class_threshold = 0.6
boxes = list()
for i in range(len(yhat)):
    # decode the output of the network
    boxes += decode_netout(yhat[i][0], anchors[i], class_threshold, input_h, input_w)
# correct the sizes of the bounding boxes for the shape of the image
correct_yolo_boxes(boxes, image_h, image_w, input_h, input_w)
# suppress non-maximal boxes
do_nms(boxes, 0.5)
# define the labels - o mesmo que coco.names
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck",
    "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench",
    "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
    "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",
    "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard",
    "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana",
    "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
    "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse",
    "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator",
    "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
# get the details of the detected objects
v_boxes, v_labels, v_scores = get_boxes(boxes, labels, class_threshold)
# summarize what we found
for i in range(len(v_boxes)):
    print(v_labels[i], v_scores[i])
# draw what we found
draw_boxes(photo_filename, v_boxes, v_labels, v_scores)
```

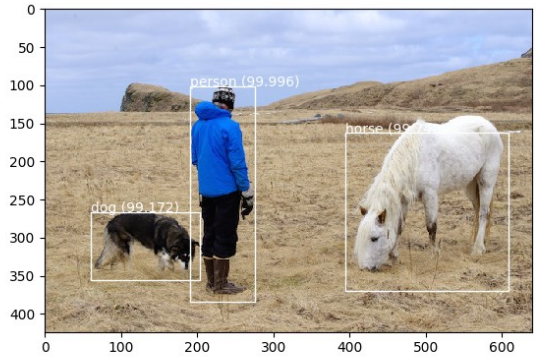
Note que o programa acima detecta somente 80 categorias de objetos listados nos labels.

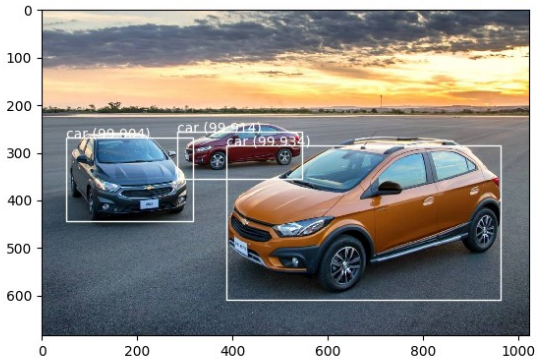
(PSI3472-2019 Aula 8 exercício 3) Execute o programa acima em algumas imagens (podem ser os do site ou da internet).

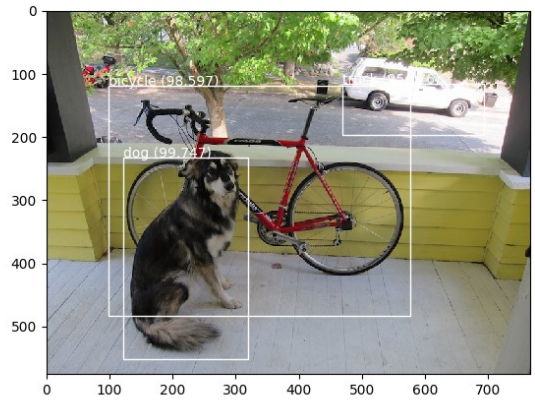
Façam:
Exercício 3 - testar Yolo.
Ou exercício 1 ou exercício 2

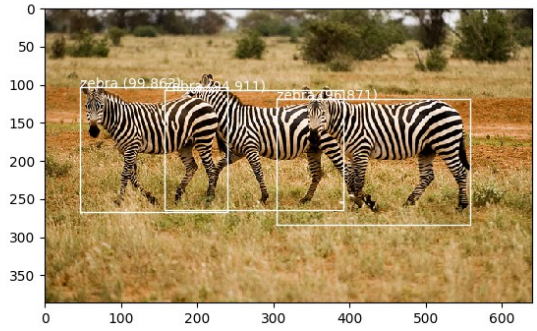
Exemplos de detecções:





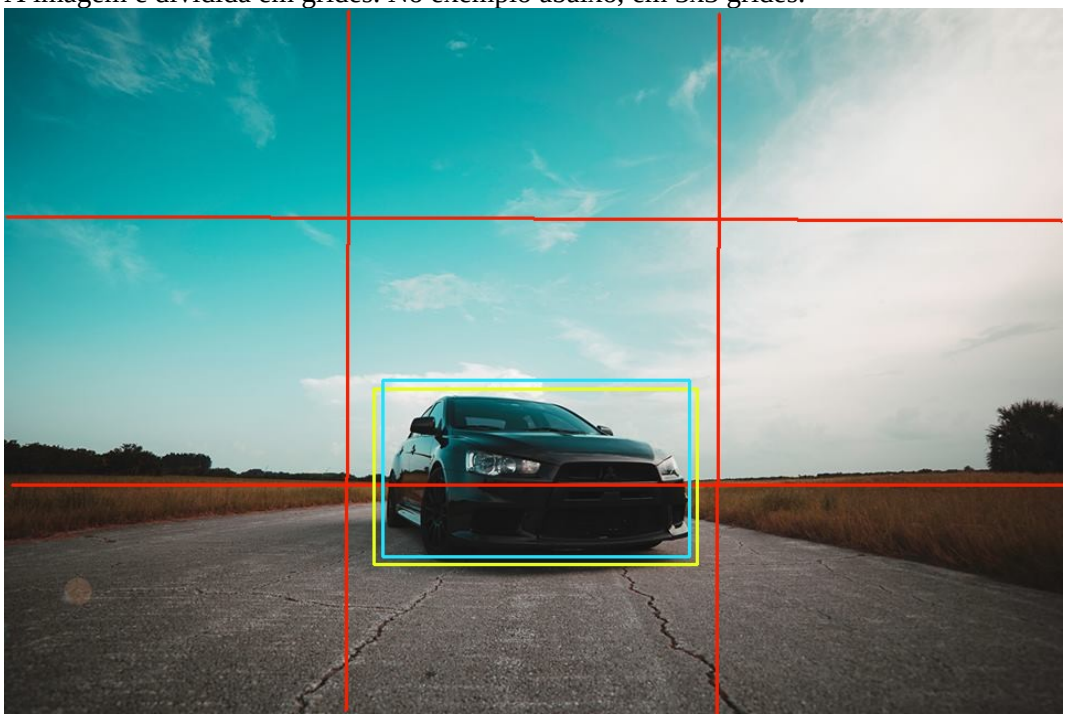






Explicação simplificada de YOLO (há detalhes que não estou explicando):

1) Digamos que queremos detectar 3 classes de objetos:
    1 = pedestre
    2 = carro
    3 = motocicleta
A imagem é dividida em grides. No exemplo abaixo, em 3x3 grides.



2) Para cada gride, faz uma predição que retorna 8 números. Isto pode ser feita fazendo uma única predição.
    y = [pc, bx, by, bh, bw, c1, c2, c3]
pc=0 se não há objeto. pc=1 se há objeto.
bx, by, bh e bw são posições x, y, height e width do bounding box do objeto. O bounding box pode sair fora do gride.
c1, c2 e c3 se referem a 3 classes de objetos.

Os grides onde não há carro:
    y = [0, ?, ?, ?, ?, ?, ?, ?]

Os dois grides onde há carro:
    y = [1, bx, by, bh, bw, 0, 1, 0]

Faz-se supressão de não-máximo, para ficar com uma única detecção de carro.