Dipartimento di Ingegneria Industriale

AUTOMATIC BALL TRACKING IN TENNIS
Dataset annotation and training

Nicola Rossi,
Veton Sulejmani

Academic year 2020/2021
Computer Vision
Prof. Nicola Conci

Indice

1 Introduction

Nowadays, one of the main fields of implementation of Computer Vision is the study of sports application, with the final goal to track relevant features, like athletes and balls motion, during a match.
In particular, the aim of this report is to build a tool able to improve the detection of a tennis ball starting from some tennis match videos.
The process will be mainly divided in two steps, which coincide with the feature extraction through an image labeling and then the consequential training process of a Neural Network, which would be able to detect the moving tennis ball in some other similar situations.

Figura 1: Ball detection with developed CNN

2 Machine Learning concept

The machine learning is a branch of artificial intelligence (AI) focused on the creation of applications which learn and improve their capability to predict the evolution of a process, through the big quantity of data given in input.
Moreover, algorithms are trained to find proper models and characteristics in order to take better decisions progressively as far as new data are processed.

The first step is the necessity to choose representative data, that will be given to machine learning model so that it apprehends the problem situation.
It is possible to divide the input data as supervised and unsupervised: the first ones are used to recall functions and classifications that final model will be able to identify, whereas for the unsupervised one, the model would assign the corresponding label by itself.
These learning data are also divided into two categories: the training data and the testing ones, whose meaning is self-explaining.

The following step consists on the decision of the algorithm to be applied, which depends mainly on the choice of supervised or unsupervised data.
There exist different ones, like the algorithm based on regression, on three approaches, clustering or associations. Finally, a neural networks consists in the definition of a layered net that has at least one hidden level, and which gives a final result through the elaboration of data along the different layers that compose it. To each output is also assigned a corresponding confidence related to a probability.
Object detectors usually exploit this principle, and, in particular, the structure of a YOLO network is similar to a normal convolutional neural network, composed of several convolutional and max pooling layers, ending with two fully connected layers.

3 YOLO Detector

The YOLO Detecor is a well known state-of-the-art, real-time object detection system, extremely flexible and widespread.
Compared to other classical approaches, YOLO represents the best choice for our purpose, as it is basically a neural convolutional network which is able to detect an object in the image by looking at it "only once".
While common approaches use a sliding window that scan every part of the image by sliding across it, YOLO uses a one stage network to perform the same analysis.
The result brings a faster output detection, keeping an high accuracy as well.

Through years, some different versions has been implemented; YOLO v3 is the one chosen for this particular development. This version returns predictions on 3 different scales, reducing the original image respectively by 32, 16 and 8

times, with the final goal to keep accuracy of prediction through different scales. YOLO v3 can work with 80 different classes already implemented in COCO dataset, which contains many different objects of common use, among which also sport (tennis) balls.
For our study, COCO dataset does not represent a good solution, and because of this a new dataset of labeled images must be created.

4 Image Labeling

In supervised machine learning world, the data labeling is a fundamental step for the correct training of neural network because it determines the quality of the model. It means that it is important the choice of the image samples given to the system and moreover the way in which this images are annotated.
In particular, there exist different types of annotation, which depend on the specific purpose of the problem that has to be solved.

- polygonal segmentation
- semantic segmentation
- 3D cuboids
- key-point and landmark
- bounding box

Among these possible choices, it was decided to perform the bounding box annotation. This one is the most commonly used type of annotation and consists on the creation of rectangular boxes that define the location of the feature of the object in the image.
Each box is represented by (x,y) coordinates in the upper-left corner and the (x,y) coordinates in the lower-right corner of the rectangle.

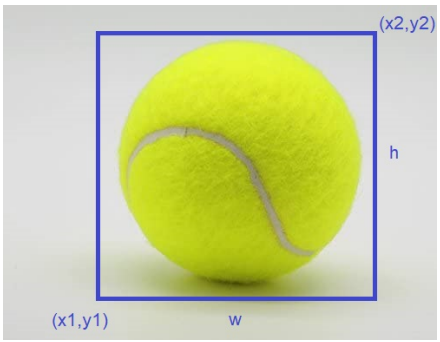

Figura 2: Bounding box labeling

Practically, it would be appropriate to design the box including not only the object, but also some border pixels. This concept is valid specially in the studied problem, where challenging situation can occur, in which the ball is not clearly visible (ball on the net, on player hand, over the court lines or on the racket).
It is possible to perform this operation using some different software tools and in this specific case *labelImg* was used.
Once the annotation is performed on the picture, it is possible to save the data related to creation of bounding boxes in different extensions file, like the one requested by YOLO. These information are stored inside a txt file, which sums up the geometrical characteristics of the box (center of rectangle and height and width), its related class and the image path.
In addition to that, the tool creates by itself another txt file with the name of all annotated classes.
As explained, differently than the other steps of neural network training, the annotation procedure still requires a lot of manual work.

5 Neural Network training

5.1 Generalities

The Artificial Neural Networks (ANN) take their name from the parallelism with the human brain.
The littlest unit, infact, is commonly called "neuron", and it can be seen as a mathematical function that takes in input a lot of data x_k, and associate to each of them a weight w_k, creating a weight vector.
This vector is what makes a neuron unique, and is continuously changed during training phase, in order to "tune" our system.
In a neural network, the output of one neuron is given as input to the following ones. The first layer is treated as "only input" because does not perform any computation, and the operation of building a network consists in choosing the number of hidden layer that characterize the net.
The other two main aspects to choose in designing a ANN are related to the type of interconnection between neurons and the learning algorithm.
The ANN, in general, presents the advantage to learn by examples, building its knowledge on dataset given during the training. The negative aspect of this type of learning is that it is not possible to know the logical path followed to obtain the output.

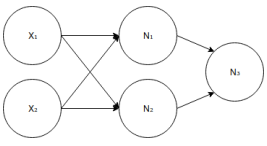

Figura 3: Basic neural network model

5.3 Results interpretation

5.3.1 Generalities

The results given by training phase of the neural network could be also represented by graphical charts.
In particular, the most common used learning curve shows the progress over the experience of a certain metric related to the training.
In other words, it is a mathematical representation of the learning process, that returns a measure of progress in the x-axis and a measure of error performances in the y-axis. This curve is called *Loss curve*, represented in figure.

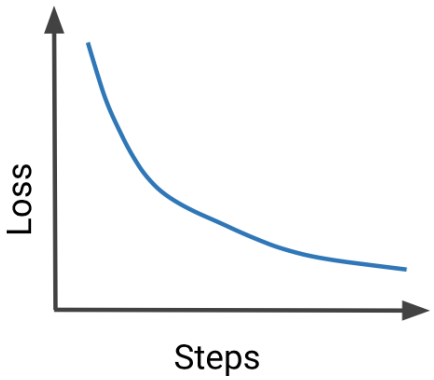

Figura 4: Loss curve model

As it can be seen from the graph, a good training should follow an asymptotic behaviour, which highlights a minimizing loss.
Due to different reasons (bad annotation, incorrect performance metric, unstable model etc.), the curve can show oscillations, saturation or "simply" cannot increase its performance with the increasing of the steps. In this case, it means that the model has not been properly trained.

5.3.2 Model results

Different types of training has been performed on this model, changing the custom dataset and some configuration parameters of the net.
In particular, in a first try it was used a dataset composed of 400 labeled images, with a *batch* parameter equal to 32 and a number of *subdivsions* equal to 8. A practical consequence of this tuning is that a larger batch number reduces the ability of the neural network to generalize the problem.
The number of batches, divided by subdivisions, also determines the number of images processed in parallel, and so influences the training time.
A second training was performed, with the aim to increase the performance of tracking in our particular situation, with a number of *batch* equal to 64, and a

number of *subdivions* equal to 16. Moreover, the custom dataset was augmented until 490 images, extracted from same input videos of the first try.

For example, the detection of tennis ball shown in figure 1, which represents a general situation of a match, was possible using the first trained model (the most "general" one), but would not be possible with the second one (specific setup model).
The loss curves obtained in the two studied models are represented in the following pictures:

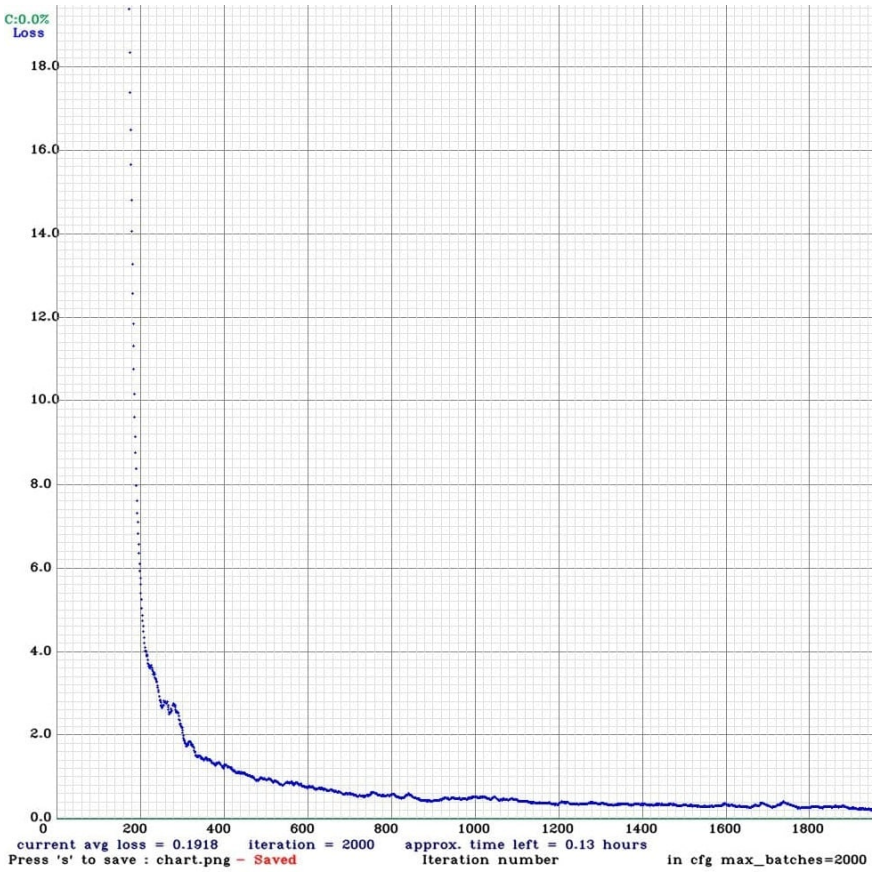

Figura 5: Loss curve of general model

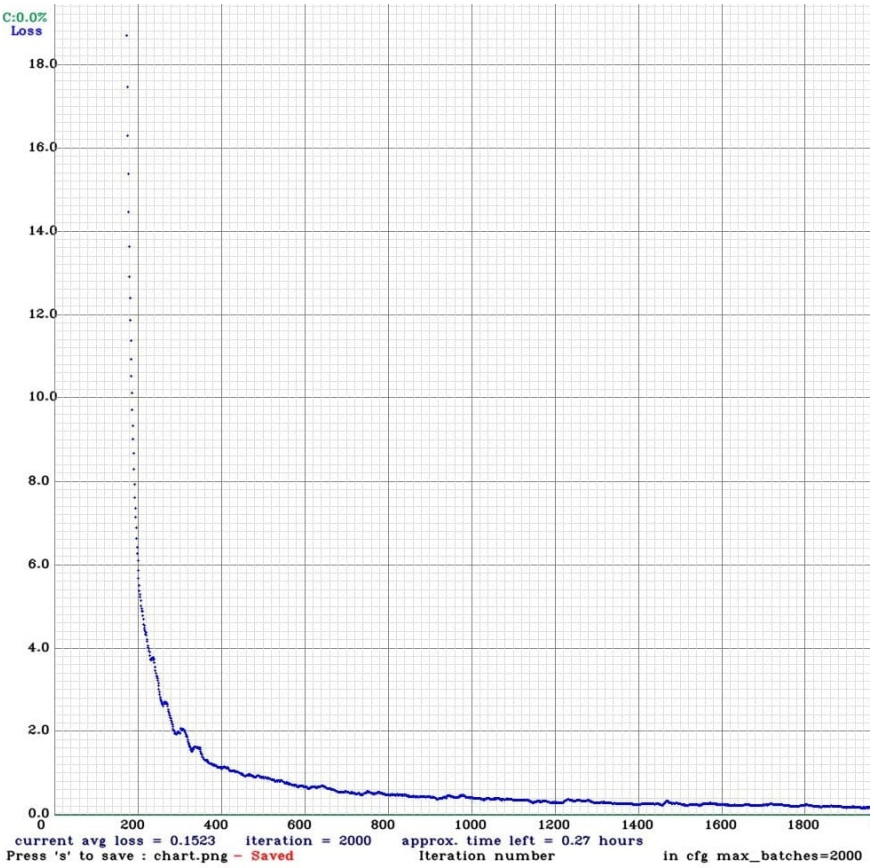

Figura 6: Loss curve of specific model

As it can be seen from previous graphs, both model loss curves follows a good behaviour and also a similar shape of the learning process.
It is also visible that the trend of the curve is still decreasing up to 2000 batches (still not saturated at this point), which means that the training performances could be further improved by tuning properly the previously explained parameters and increasing the number of training samples.

5.4 Python code detector

The final phase of the process consists in creating a piece of code able to build, once the tennis ball is detected in a new input image, the corresponding bounding box and show also the associated level of confidence (in range 0-1%).
In order to do that, the Python code is reported below, with a line-by-line comment:

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

# load yolo models
net = cv2.dnn.readNetFromDarknet('yolov3_custom.cfg', 'yolov3_custom_final.weights')
# create list of classes
classes = []
# trained with our dataset
with open('classes.names', 'r') as f:
    classes = [line.strip() for line in f.readlines()]
# video path
cap = cv2.VideoCapture('test_def.mp4')

# image path (for image detection)
#my_img = cv2.imread('test1.jpg') #provide path of images
while True:
    #extract each frame
    _, my_img = cap.read()
    ht, wt, _ = my_img.shape #weight, height and rgb channel of our image
    #divide every single in 255 -> color range from 0 to 255 from 0 to 1
    #we have defined shape 416x416 of the blob
    blob = cv2.dnn.blobFromImage(my_img, 1/255, (416, 416), (0, 0, 0), swapRB = True, crop = False)
    net.setInput(blob)
    last_layer = net.getUnconnectedOutLayersNames() #define last layer of neural network
    layer_out = net.forward(last_layer) #we want to get output from that last layer
    #first 5 elements talks about bounding box
    #and after these we will have the number of classes and their probabilities

    #define some basic variables
    boxes = []
    confidences = []
    class_ids = []
```

```python
    #use for loop to figure out bounding boxes, confidence and class class_ids
    for output in layer_out:
        for detection in output:
            #probabilities come after 5th element
            score = detection[5:]
            class_id = np.argmax(score) #get index of class associated to highest probability
            confidence = score[class_id] #returns affidability value of current prediciton

            if confidence > 0.0: #set this threshold to interested confidence (we put 0 to show all
            #bounding boxes geometry along video)
                center_x = int(detection[0] * wt)
                center_y = int(detection[1] * ht)
                w = int(detection[2]* wt)
                h = int(detection[3]* ht)
                x = int(center_x - w/2)
                y = int(center_y - h/2)

                #let's append our boxes, confidences and class id
                boxes.append([x, y, w, h])
                confidences.append((float(confidence)))
                class_ids.append(class_id)
```

```
    #NMS removes redundant overlapped boxes
    #if you have a large value, you are enforcing two boxes to have a very high
    #overlap (which is usually not the case)
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    #assign a font
    font = cv2.FONT_HERSHEY_PLAIN
    colors = np.random.uniform(0, 255, size=(len(boxes), 3))
    #use flatten command because "indexes" is list of lists
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]]) #assign to label the corresponding class id
            confidence = str(round(confidences[i], 2)) #approximate confidence
            color = colors[i]
            cv2.rectangle(my_img, (x, y), (x+h, y+h), color, 2) #image, initial coordinates and final
            # ones, color and size
            cv2.putText(my_img, label + " "+confidence, (x, y+20), font, 2, (0,0,0), 2)

    cv2.imshow('img',my_img)
    key = cv2.waitKey(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

6 Conclusions

The evaluation step among which the process was finally tested is the capability to detect the tennis ball in images and videos. In particular, analyzing the tracking of the ball in different videos (some of them representing the same setup situation from where training samples has been extracted and others from completely different setup views), it was observed that in most of the frames the tracker is able to detect the ball. The tracking detection of the ball in the half court closer to camera results pretty convincing, whereas, in the further one, the ball is more frequently confused with similar shapes or reflection of other objects.

Figura 7: Hit detection in indoor court

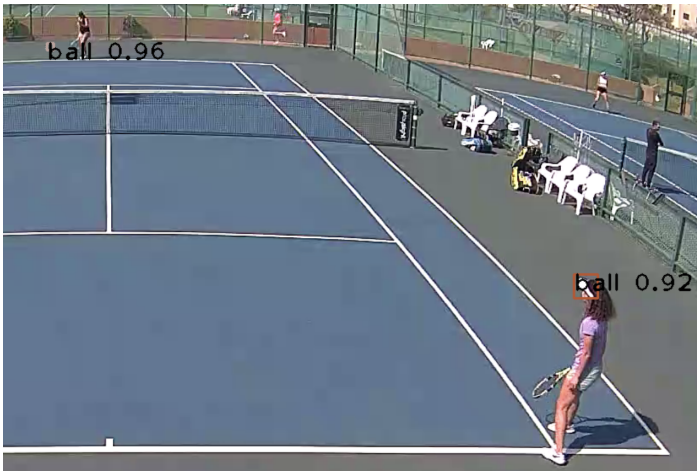

Figura 8: False positive detection in outdoor court

It was also observed that the detector performances strongly depend on the camera setup and environmental conditions (illumination conditions or similar colors and contrast between background and tennis ball).
In the first situation, the indoor court, the detector gives quite satisfactory results corresponding to an high confidence on ball prediction in most of the frames and in both sides of the court. The second picture, instead, shows an outdoor setup, in which the detector does not work so well, especially in some challenging situations.
In this case, other than miss detection, also some false positive (the hat is detected as a ball) occurs.
To conclude, this tracker could be a good solution for the specific application mainly in tennis indoor halls, thanks to the higher contrast between tennis ball and wall surfaces, which makes the tracking process easier and much more robust.

7 References

- 1. *"https://pjreddie.com/darknet/yolo/"*
- 2. *"https://www.spindox.it/it/blog/yolo-riconoscimento-oggetti-real-time/"*
- 3. *"https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1/"*
- 4. *"https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73"*
- 5. *"https://www.baeldung.com/cs/learning-curve-ml"*
- 6. *"https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e"*
- 7. *"https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu"*