Object Detection Web App Using Flask Framework, COCO Dataset and Tensorflow

Adeel (BCSF17A556)

Abid Shafique (BCSF17A521)

Muhammad Naveed (BCSF17A534)

Muhammad Tayyab (BCSF17A512)

Date: 24-01-2021

Introduction

Human visual system is the most powerful system. Human eye is fast, accurate and can perform complex tasks. It can detect, identify and localize objects present in an image. Here, we need to understand terms such as object detection, object recognition, and object localization.

Object recognition *[1]* refers to a collection of related tasks for identifying objects in digital photographs. Region-based Convolutional Neural Networks *[2]* is a family of techniques for addressing object localization and recognition tasks, designed for model performance.

The availability of large sets of data, faster GPUs *[3]*, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. So, we can distinguish between these three computer vision tasks with this example:

Object Recognition: This is done through, verifying the presence of objects in an image.

Input: An image, which consists of one or more objects, such as a photograph.

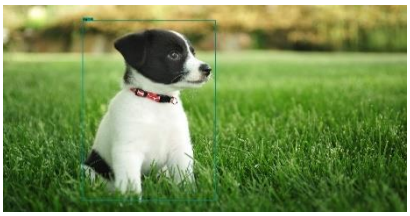
Output: Yes or No.

Object Localization *[4]**:*** This is done through, locating the presence of objects in an image and indicate their location with a bounding box.

Input: An image, which consists of one or more objects, such as a photograph.

Output: One or more bounding boxes (e.g. defined by a point, width, and height).

Object Detection *[5]**:*** This is done through, locating the presence of objects with a bounding box and types or classes of the located objects in an image.



Input: An image, which consists of one or more objects, such as a photograph.

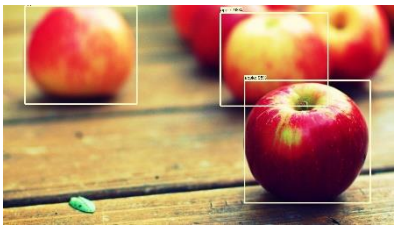
Output: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.

The main goal of our project is to detect, recognize and localize an object present in an image using Common Object in Context *[6]* model with the help of TensorFlow *[7]* and Flask Web Framework *[8]*.

Literature Review

Context Object in Context Dataset: Context Object in Context Dataset *[6]* is a popular dataset for computer vision. Format of this dataset is automatically understood by advanced neural network libraries, e.g. Facebook's Detectron2 *[9]*. Understanding how this dataset is represented will help with using and modifying the existing datasets and with creating the custom ones. This model helps us to detect the objects. Model should get bounding boxes for objects,






i.e. return list of object classes and coordinates of rectangles around them; objects (also called “things”) are discrete, separate objects, often with parts, like humans and cars; the official dataset for this task also contains additional data for object segmentation (see below)

In computer vision, object detection has tremendous usage, e.g. for self-driving vehicles (detection of people and other vehicles), AI-based security (human detection and/or segmentation) and object re-identification (object segmentation or removing background with stuff segmentation helps with checking object identity).
TensorFlow: TensorFlow *[7]* is a powerful data flow-oriented machine learning library created by the Brain Team of Google and made open source in 2015. It is designed to be easy to use and widely applicable on both numeric and neural network-oriented problems as well as other domains.

Basically, TensorFlow *[7]* is a low-level toolkit for doing complicated math, and it targets researchers who know what they’re doing to build experimental learning architectures to play around with them and to turn them into running software.

It can be thought of as a programming system in which you represent computations as graphs. Nodes in the graph represent math operations, and the edges represent multidimensional data arrays (tensors *[10]*) communicated between them.

Flask Web Framework: Flask *[8]* is a small and lightweight Python *[11]* web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file.

Implementation Details

To understand the implementation, first we have to understand about Faster R-CNN *[2]*:

Faster R-CNN: Faster R-CNN stands for Faster Region-based Convolutional Neural Networks *[2]*, a network that does object detection. Its working is as follow:

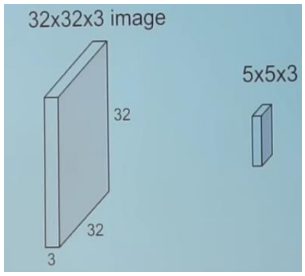
Step 1: Run the image through a CNN *[12]* to get a Feature Map.

Step 2: Run the Activation Map through a separate network, called the Region Proposal Network (RPN) that outputs interesting boxes/regions.

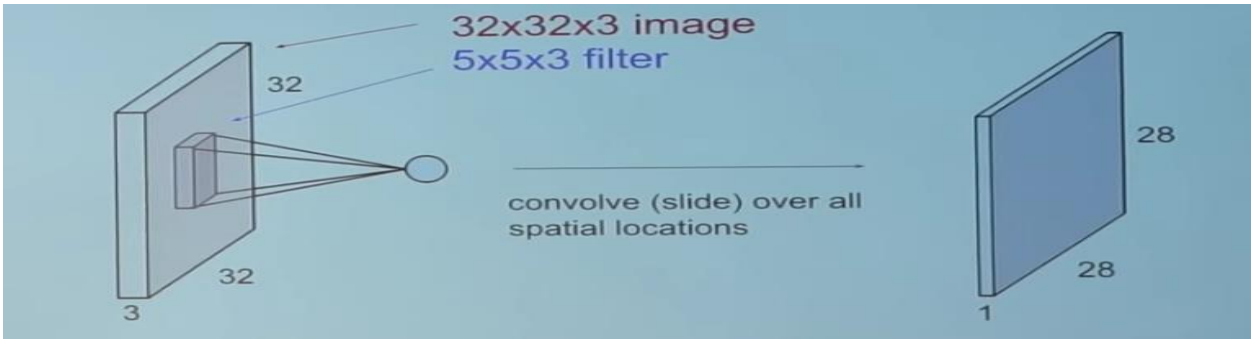
Step 3: For the interesting boxes/regions from RPN use several fully connected layer to output class + Bounding Box coordinates.

To move further in Faster R-CNN *[2]* we should be clear in our understanding on CNN *[12]*.

Working of CNN: CNN stands for Convolutional Neural Networks *[2]*. In deep learning, a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics. Unlike neural networks, the input is a vector with a multi-channeled image (three channeled- RGB *[13]* in this case).



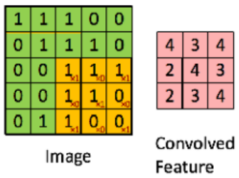
Convolution *[14]* is the first layer to extract features from an input image. It preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

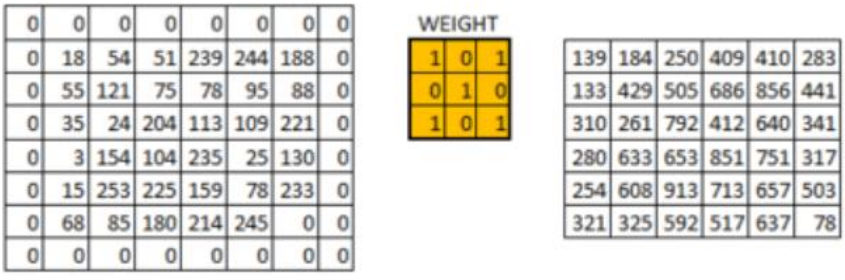


From the above image, we can observe that for our input of 32*32*3 we took a filter of 5*5*3 and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image. The output results with an image of size 28*28*1 e.g. consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below.

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “Feature Map” as output shown in below with a stride of 1.

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. If we increase the stride value, the size of image keeps on reducing, padding with zeros across it solves this problem. From the below image we can observe the size of image is retained after padding zeros with stride 1.





ReLU stands for Rectified Linear Unit *[15]* for a non-linear operation. Its purpose is to introduce non-linearity which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers. Pooling is done for the sole purpose of reducing the spatial size of the image. Pooling *[12]* is done independently on each depth dimension;

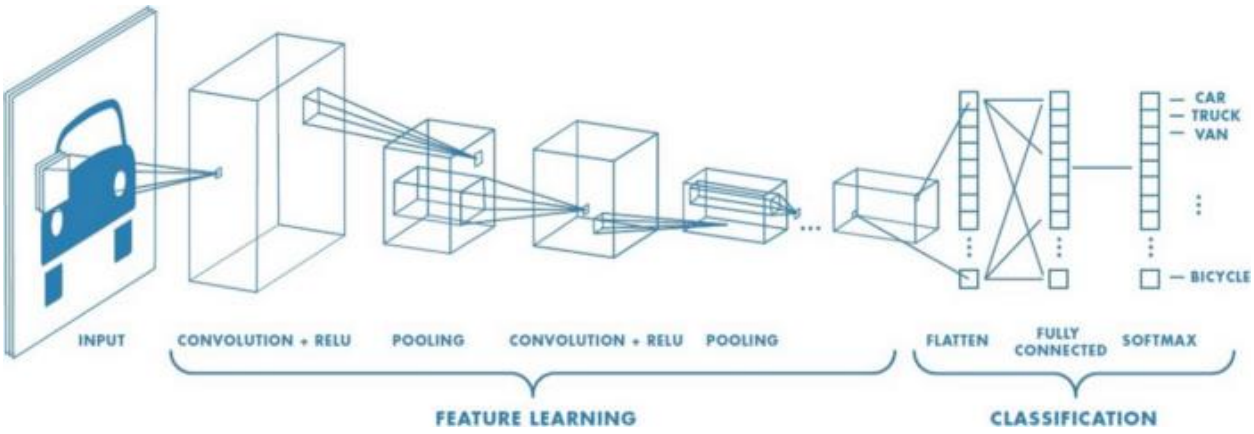
therefore the depth of the image remains unchanged. The most common form of pooling layer generally applied is the max pooling, e.g.

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

792	856
913	851

Here we have taken stride as 2, while pooling size also as 2. The max operation is applied to each depth dimension of the convolved output. As you can see, the 4*4 convolved output has become 2*2 after the max pooling operation.

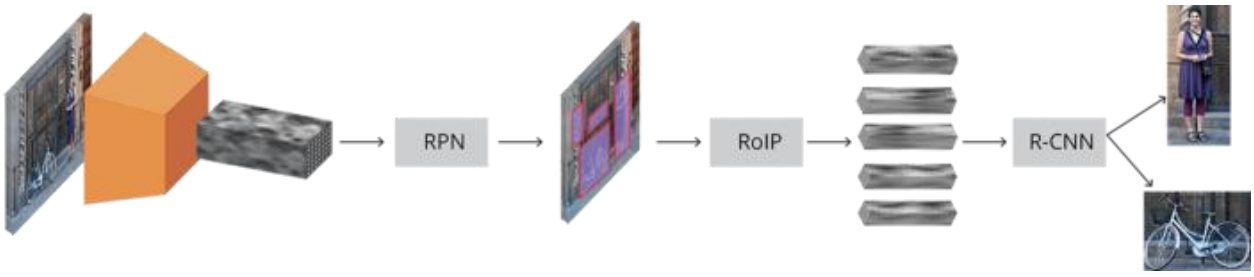
The below figure is a complete flow of CNN *[12]* to process an input image and classifies the objects based on values.



The feature map matrix after pooling layer will be flattened as vector. With the fully connected layers, we combined these features together to create a model. Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class. The output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete, the backpropagation begins to update the weight and biases for error and loss reduction.

Now after getting a feature map from the CNN *[12]*, we need to pass it to Region Proposal Network (RPN), to find interesting regions. It returns regions of interest, the loss regarding whether it found an object and the loss regarding the location of the object.

Following is the high-level architecture for Faster-RCNN:



Once we got the features for the input image from CNN, generation of region proposals (Anchors/Bounding Box) can be done with Region Proposal Network (RPN) layer. The predicted region proposals are then reshaped using a Region

of Interest Pooling (RoIP) layer, which is then used to classify the image within the proposed region and predict the offset values with R-CNN for the bounding boxes.

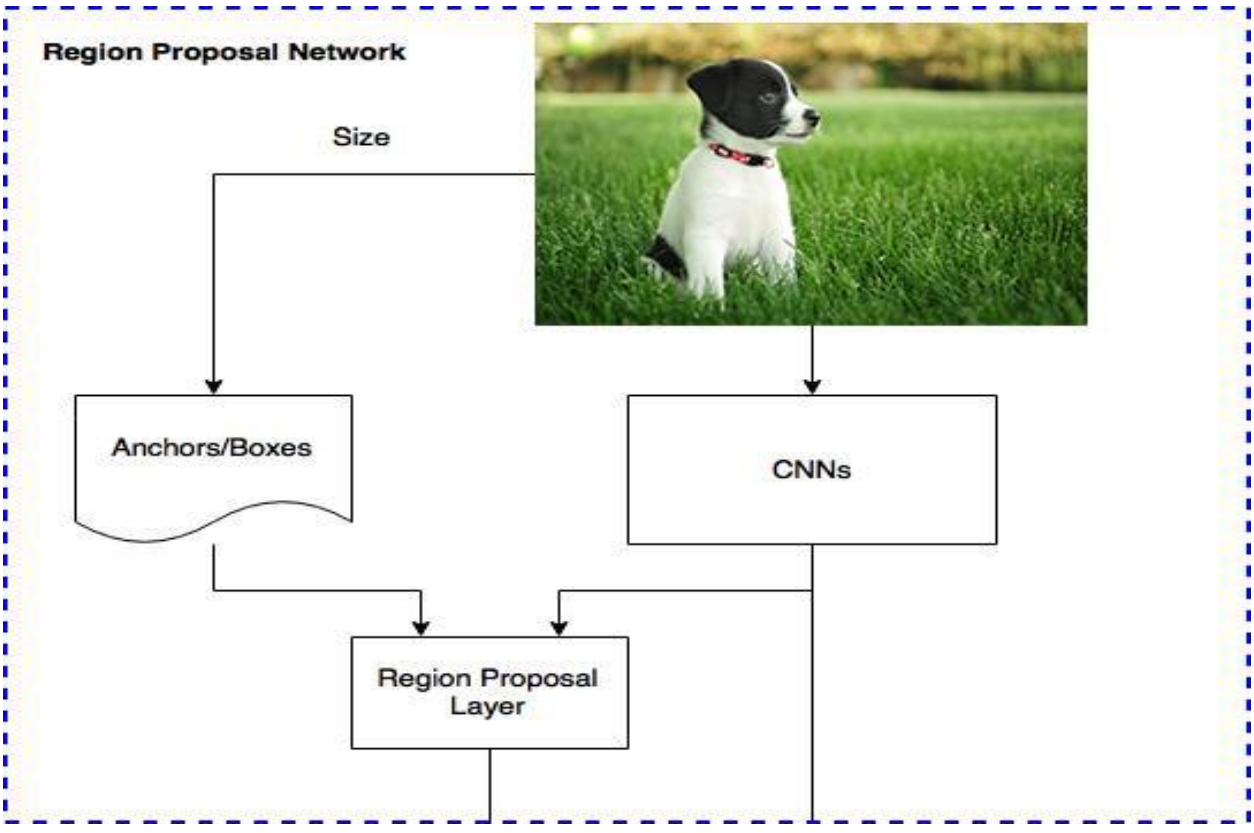
It's a method where we use selective search to extract just 2000 regions from the image (region proposals). Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions. These 2000 candidate region proposals are warped into a square and fed into a CNN that produces a 4096-dimensional feature vector as output. The CNN acts as a feature extractor, the output dense layer consists of the features extracted from the image, and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box. For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could've been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal.

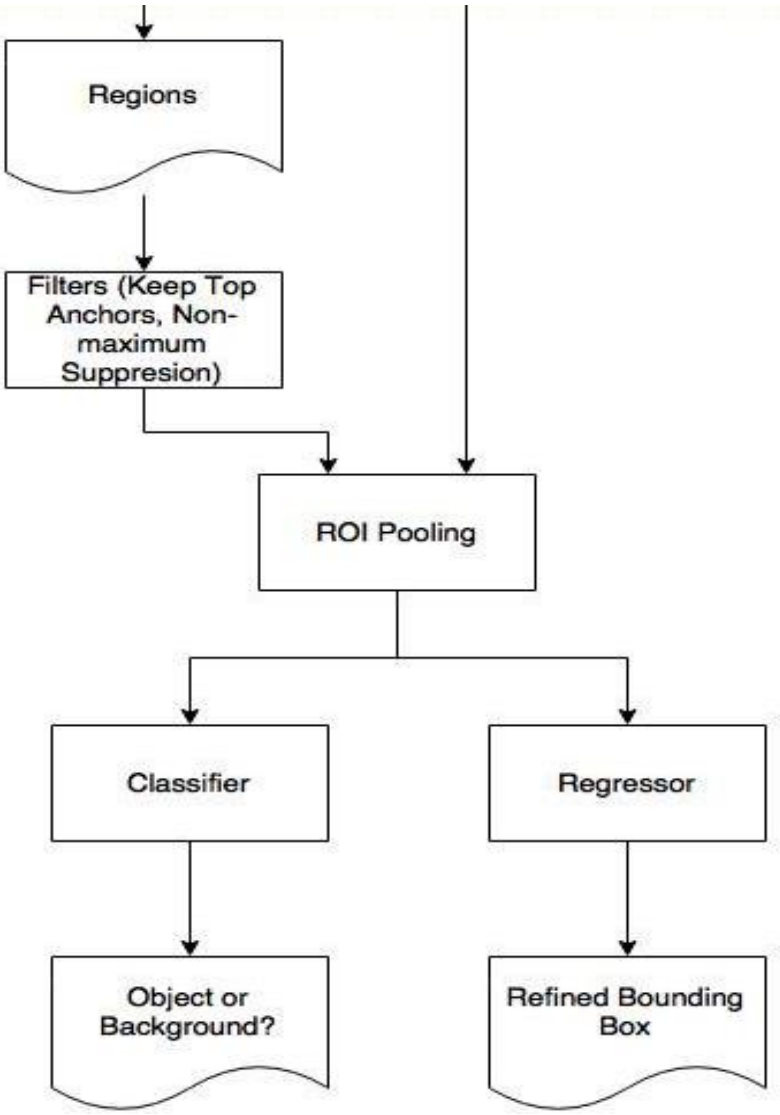
Instead of feeding the region proposals to the CNN *[12]*, we feed the input image to the CNN *[12]* to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoIP layer, we reshape them into a fixed size so that it can be fed into a fully connected layer. From the RoI *[16]* feature vector, we use a softmax layer to predict the class of the proposed region and the offset values for the bounding box.

The reason "Fast R-CNN" is faster than R-CNN *[2]* is that you don't have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it.

Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.

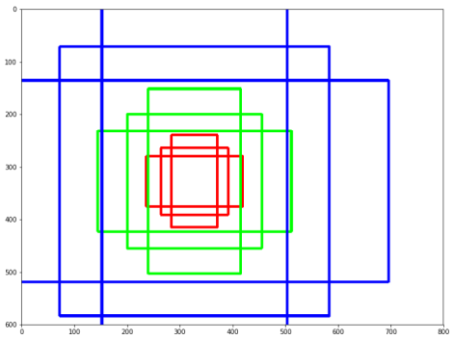
Architecture of Faster R-CNN:





The above architecture will explain the sequence of operations performed once an image got its feature matrix from CNN layer.

Working of RPN: After getting a set of convolutional feature maps (feature matrix) on the last convolutional layer a sliding window is run spatially on these feature maps. The size of sliding window is n×n (here 3×3). For each sliding window, a set of 9 anchors are generated which all have the same center (x_a, y_a) but with 3 different aspect ratios and 3 different scales as shown below.



Let's look closer:

1. Three colors represent three scales or sizes: 128x128, 256x256 and 512x512.

2. Let's single out the red boxes/anchors. The three boxes have height width ratios 1:1, 1:2 and 2:1 respectively.

These anchors work well for the COCO *[6]* dataset. However, you have the freedom to design different kinds of anchors/boxes. For example, you are designing a network to count passengers/pedestrians; you may not need to consider the very short, very big, or square boxes. A neat set of anchors may increase the speed as well as the accuracy.

Furthermore, for each of these anchors, a value $p*$ is computed which indicated how much these anchors overlap with the ground-truth bounding boxes.

$$IoU = \frac{A \cap Gt}{A \cup Gt} \begin{cases} > 0.7 = \text{object} \\ < 0.3 = \text{not object} \end{cases}$$

where ***IoU*** is intersection over union and is defined below:

$$IoU = \frac{Anchor \cap GT\,Box}{Anchor \cup GT\,Box}$$

Finally, the 3×3 spatial features extracted from those convolution feature maps are fed to a smaller network, which has two tasks: classification and regression. The output of regressor determines a predicted bounding-box (x, y, w, h), the output of classification sub-network is a probability p indicating whether the predicted box contains an object (1) or it is from background (0 for no object).

Working of ROI Pooling Layer: After RPN, we get proposed regions with different sizes. Different sized regions mean different sized CNN feature maps. It is not easy to make an efficient structure to work on features with different sizes. ROI *[16]* Pooling can simplify the problem by reducing the feature maps into the same size.

The result is that from a list of rectangles with different sizes we can quickly get a list of corresponding feature maps with a fixed size

Final layer R-CNN: Region-based convolutional neural network (R-CNN) *[2]* is the final step in Faster R-CNN's pipeline. After getting a convolutional feature map from the image, using it to get object proposals with the RPN and finally extracting features for each of those proposals (via RoI Pooling), we finally need to use these features for classification. R-CNN tries to mimic the final stages of classification CNNs where a fully connected layer is used to output a score for each possible object class. R-CNN *[2]* has two different goals:

i. To classify proposals into one of the classes, plus a background class (for removing bad proposals).

ii. To better adjust the bounding box for the proposal according to the predicted class.

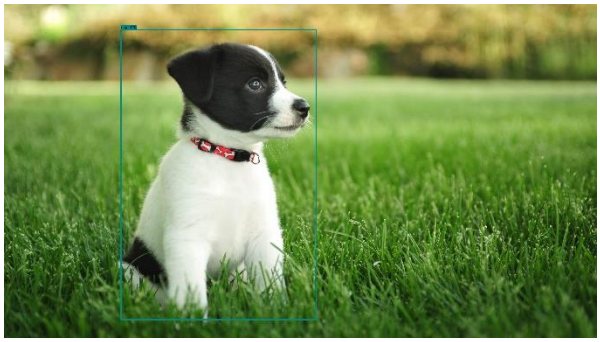
Train some sample images of Objects, so that out model should recognize the Object from the given image.

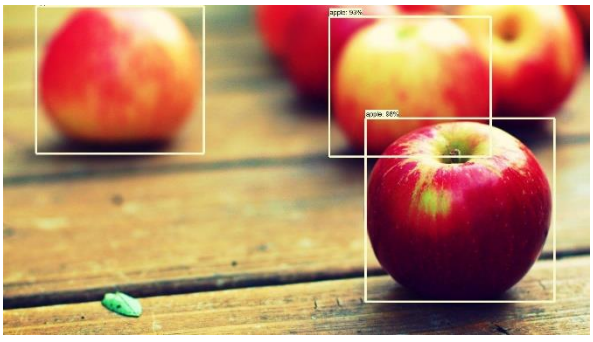
Implementation Flask Web App: After training, now we need to implement the flask app, generally a website contain frontend and backend *[17]*. So, let's first talk about frontend *[17]*, first of all an HTML *[18]* view is required in that contain at least a form with a Submit button and a field for File.

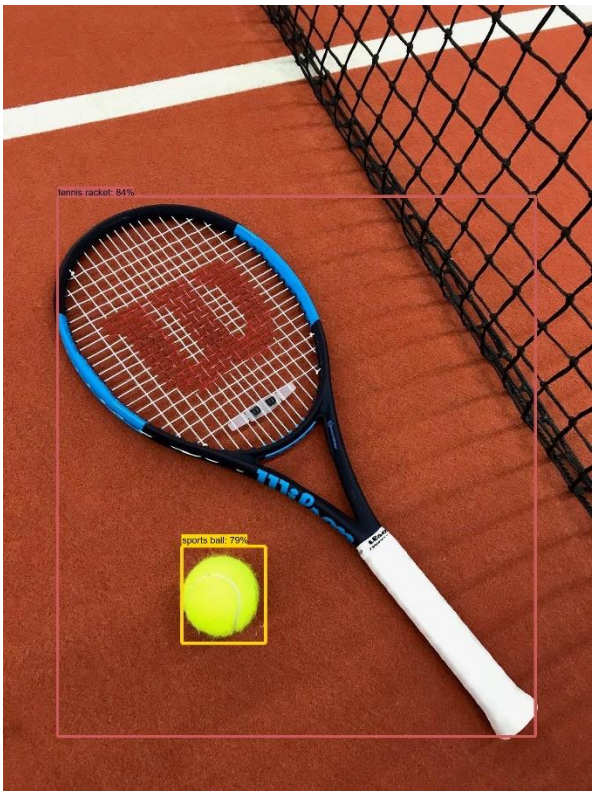
Another HTML *[18]* could be made be for the result; otherwise, it must be shown on browser directly. These views must be placed in a *templates* and this folder may contain another folder shared for external CSS *[19]* styles, javascript *[20]* files, icons, fonts, etc.

Now to process an image using tenserflow to detect objects in it, let's move to backend, generally for a simple flask app it's compulsory to create a python file that initialize it. For this, create a file named *app.py*, it's just a naming convention, it may have any name but it's a good practice to name it as app.py. In this file, we must have to write a POST *[21]* method route along with flask initialization. By POST *[21]*, we can take an image and process it using trained model with the helped of tenserflow. After that host app on 0.0.0.0:5000 to run it on https://localhost:5000.

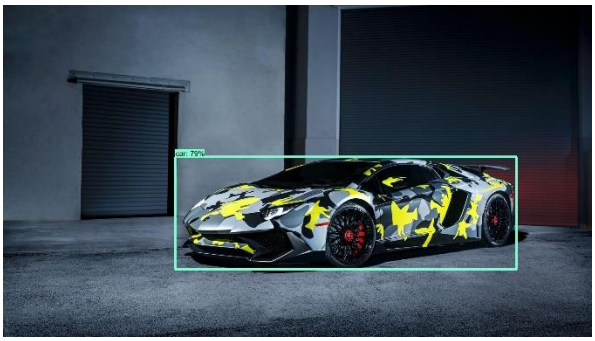
Results






tennis racket: 84%
sports ball: 79%

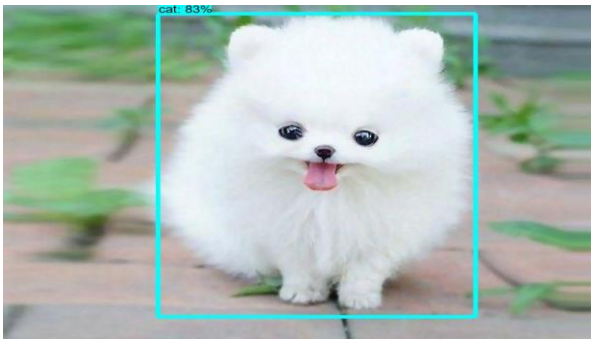

cat: 91%


car: 79%

Conclusion

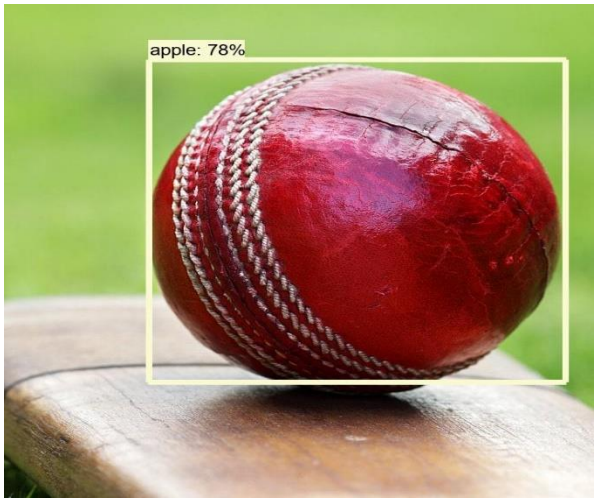
In this project, we have developed object detection web app, which will take and input image and locate the object presents in the image with bounding boxes. It will assign a label according to the type of objects it detects. For example, if it detects horse in an image it makes a bounding box around the horse, label it a horse, and will show the accuracy of the object as show in above results. It works fine in for many cases but not a perfect app it has some own drawbacks and sometimes does not work as we expect. It can only detect those objects whose dataset is trained, sometimes it ignores some objects or sometimes it mixed up some objects with each other. Some of unexpected results are shown below:

In image following image, a Dog exists but app labeling it as a Cat:



In image following image, a soccer team exists but app is not detecting some person from them:

In image following image, a red Sports Ball exists but app labeling it as an Apple:



References

[1]: https://en.wikipedia.org/wiki/Convolutional_neural_network

[2]: https://en.wikipedia.org/wiki/Region_Based_Convolutional_Neural_Networks

[3]: https://en.wikipedia.org/wiki/Graphics_processing_unit

[4]: https://machinelearningmastery.com/object-recognition-with-deep-learning/#:~:text=Object%20localization%20refers%20to%20identifying,more%20objects%20in%20an%20image.

[5]: https://en.wikipedia.org/wiki/Object_detection

[6]: https://en.wikipedia.org/wiki/TensorFlow

[7]: https://arxiv.org/pdf/1405.0312.pdf

[8]: https://en.wikipedia.org/wiki/Flask_(web_framework)

[9]: https://ai.facebook.com/tools/detectron2

[10]: https://en.wikipedia.org/wiki/Tensor

[11]: https://en.wikipedia.org/wiki/Python_(programming_language)

[12]: https://en.wikipedia.org/wiki/Convolutional_neural_network

[13]: https://en.wikipedia.org/wiki/RGB_color_model

[14]: https://en.wikipedia.org/wiki/Convolution

[15]: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[16]: https://en.m.wikipedia.org/wiki/Region_of_interest

[17]: https://en.wikipedia.org/wiki/Front_end_and_back_end

[18]: https://en.wikipedia.org/wiki/HTML

[19]: https://en.wikipedia.org/wiki/CSS

[20]: https://en.wikipedia.org/wiki/JavaScript

[21]:https://en.wikipedia.org/wiki/POST_(HTTP)#:~:text=In%20computing%2C%20POST%20is%20a,submitting%20a%20completed%20web%20form.