Object Detection using COCO SSD & ml5.js

# Introduction

In today's world we can find severals of machine learning or deep learning applications which involve object detection using the digital camera. It helps the computer to understand the content and the context of the image. This also helps those tiny silicon chips to read the text captured on the images. This open ups a lot's of possibilities to us. Amazon Go, the smartest grocery store without any shopkeeper or human cashier, is one of the greatest example of this amazing technology. Car overspeed fine system also use the same technology to get the car number from it's number plate via those images captured by the sensors and the system. It saves lot's of human effort and with the performance it also increase the efficiency and accuracy. This all is the game of finding pattern from a bunch of numeric matrix. We human are so efficient with patterns but computers are not so comfortable with patterns, they understand numbers, hex codes, zeros and ones. In this current era, machine learning and deep learning is breaking this barrier and helping out computers to see through those numbers and feel those hidden patterns so that they can work on those new information and make it’s own decision without any human interaction.

# Deep learning & CNN

Deep learning is the most powerful tool in the toolbox of Artificial Intelligence. It mimic the real human brain architecture, the neurone, perceptron, axion. Like the real human brain it also has lots of neurone, and a network of those interconnected neurone, which is known as Artificial Neural Network or ANN in short. It use a complex back propagation technique for training purpose. With little bit of modification it can be used for both kind of task, weather it is regression or classification. The only downside is that unlike the machine learning algorithms, it is super complex, gigantic beast which required lots of computing power and huge amount of train, testing and validating data. Still, after all those complexity and heavy requirements the accuracy and flexibility make it worth and on the top of the line.

Although of it's accuracy and flexibility, till now we just saw how much better it is than the traditional machine learning methods, not more than that; but there is a 'but'. ANN is just the foundation of the whole big magical empire. The main game starts now. There is a special kind of modified artificial neural networks which can go beyond the imagination and can do staffs which we saw in science friction movies earlier. It's CNN or Convolution Neural Network. This state of the art algorithm can find out patterns or features from the images like we human do. It takes a 2D images, maybe it's a colour image or grayscale image, and run a small filter window through out the whole image step by step and after applying the filter all over the image, the highlighted features are pass it to the next layer. The first few CNN layers can only highlight the simple things like, horizontal line, vertical line, tilted line etc. As the layer goes deeper it become able to detect much more complex shapes like, a circle, rectangle etc. After properly highlight the important features in the image if anyone want then he/she can get that feature map as output, but to detect or predict something, most of the time we flat this 2D image to a single dimension array and then use normal ANN layers and do classification or regression, which is suitable for us.

# Object Detection & CNN

Object Detection is to take an image, check weather any known object is present in that image or not, if any known object found then classified that object and labeled it and most importantly mark the position the object is located in the image. There may be multiple objects present on the single image with different size and shape of same of multiple classes, or there may be no object present on the image. Now it maybe confusing to you that it’s nor a classification problem, neither a regression problem, how can we solve this kind of problem. Still in the deep of your mind you most probably trying to link it with the classification problem. The good news is that you are right; it’s a kind of classification problem that have to solve using a different complex approach.

Let’s talk about the most general object detection technique, how it’s work, what processes are involved in it and many other things.

First of all dataset. Like any other dataset, there is a training images and labels, but unlike any other labels, these labels not only store a single object label for a single image but one or more objects names with their corresponding position in the single image, or with the ground truth boxes.

Next the detection mechanism. For that the model create a bunch of boxes of different size and aspect ratio in different places on that image. Now the SSD model with check for those boxes which are overlapping with the ground truth boxes or the labels. Then it IoU or Intersection over Union for the heights overlapping box, and this box is shown as the output.

Now for matching the detector use severals of boxes with overlapping over 50% and compared with the labeled ones for better accuracy.

Let’s talk about the role of CNN in this whole process. Obviously the silicon chip can’t take two photo side by side and compare them to tell weather they are same or not. It must have the extract some features form it, and have to compare those feature to tell weather those two images are same or not. In this case CNN is a champ in feature extraction, and with it’s parallel running capabilities it can do it much faster than any other serial machine learning algorithm. Also the feature map of CNN helps us to find the accurate location of the object in the picture.

The SSD model we used here also use a quite popular CNN network for this, which is *VGG-16*.

# SSD Model & VGG-16

SSD or Single Shot Detector is created for accurate live object detection with the help of region based Faster R-CNN. Although it’s accuracy is quite fascinating but it has some payback. The frame per second is limited to only 7 FPS, which is far behind than it’s closest competitor YOLO with astonishing 45 FPS. Still the FPS can be recovered by eliminating the R-CNN and reducing the input resolution.

Let’s talk about the inner mechanism. As we discuss earlier, in the first stage we have to extract features from the image in order to enable the computer to work on it. For that SSD use VGG-16 network to generate the feature maps. In the next stage it use the Conv4\_3 layer to detect objects in the image. It use different size of feature maps. For large scale object detection it use low resolution feature maps and for small scale object it use high resolution feature maps. Here is the list of filter map sizes used in SSD - 38x38, 19x19, 10x10, 5x5, 3x3 and finally 1x1.

It use total 8732 boxes to detect objects with a great confidence. To detect the same objet of different sizes and aspect ratio it combines multi-scale features maps, and to eliminate the duplicate predictions SSD use non-maximum suppression (NMS). If the previous prediction of the same class object has higher IoU score then it drop the current prediction and continues with the previous one, other wise the previous one is dropped and continues with the current one. Finally, among those 8732 boxes it takes only top 200 boxes for prediction per image.

# Implementation

The whole project is based on a single JS library, ml5. Little bit of HTML & CSS is used to create the web interface. Let’s have an overview of the application. It’s a single page static web app, which will use the default webcam as the input and will show the output with the borders and labels on the whole canvas.

At the head section of the HTML we import the ml5 using a CDN link. There is no design related HTML tag on the body, just only a script tag, with all the main code.

At the top of our script we declare some important variables and some constants of display sizes. Following that we create a init function with will initialise the application on the page load.

const init = async () => {

vdo = await getVideo();

modelDetector = await ml5.objectDetector(

‘cocossd’,

startDetecting

)

cnv = canvasConstructor(width, height);

ctx = cnv.getContext('2d');

}

window.onload = init;

Here is getting the video input source using the getVideo() function which we will talk about later in this section. Then it’s load the pre trained coco ssd model with a callback function. We also create a canvas for the output using our canvasConstructor function, which we will discuss later. It stores all of its result to those global variable which we declared at the top of the script. Then we add an javascript event to call this init function on the page load.

Let’s have a look over the canvasConstructor function, which is responsible for creating the output screen.

function canvasConstructor(ww, hh) {

const cnv = document.createElement("canvas");

cnv.width = ww;

cnv.height = hh;

document.body.appendChild(cnv);

return cnv;

}

This function takes to parameter as input, which is width and height of the canvas and create a canvas element of that given size and then it insert this canvas to the document body and also return the canvas object for further operations.

Now it’s time to see how we are getting video input from the webcam. For this we are using our getVideo() function. Let’s understand what’s going on under the hood.

async function getVideo() {

const vdo = document.createElement('video');

vdo.style.display = "none";

vdo.width = width;

vdo.height = height;

document.body.appendChild(vdo);

const vdoInp = await navigator.mediaDevices.getUserMedia({

video: {width, height}

})

vdo.srcObject = vdoInp;

vdo.play();

return vdo;

}

This function is creating a video element of the window height and width with no visibility and inserting it to the document body. Then using the media devices it’s getting the default webcam with our predefined output size. In the next step it’s starting the video and returning it.

Before getting to our main detect function let’s have a look on draw function which will draw boundary boxes and labels on the canvas.

const draw = () => {

ctx.fillStyle = "#222"

ctx.fillRect(0, 0, width, height);

ctx.drawImage(vdo, 0, 0);

for (let i = 0; i < detectedObjects.length; i++) {

ctx.font = "22px Georgia";

ctx.fillStyle = "blue";

ctx.fillText(detectedObjects[i].label, detectedObjects[i].x + 6, detectedObjects[i].y + 12);

ctx.beginPath();

ctx.rect(detectedObjects[i].x, detectedObjects[i].y, detectedObjects[i].width, detectedObjects[i].height);

ctx.strokeStyle = "blue";

ctx.stroke();

ctx.closePath();

}

}

This will fill the whole background with white and for each detected objects it will insert a text item with the label on the at the beginning of that object on the image. Then it will draw a rectangle of the detected width and height on will place it over the object on the image.

Now, finally in the detect function we are giving the video input source to the model and storing the result on the global variable and asking the draw function to draw the border on the image, and then again the detect function calling it self recursively.

const detect = () => {

modelDetector.detect(vdo, function (err, output) {

if (err) return console.log(err);

detectedObjects = output;

if (detectedObjects) draw();

detect();

});

}