REAL TIME OBJECT DETECTION

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

AJEYA DEY (34200115003)

AMLAN BOSE (34200115004)

ANAMITRA CHOWDHURY (34200115005)

ORIJITA ADHIKARY (34200115032)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FUTURE INSTITUTE OF TECHNOLOGY

2019

REAL TIME OBJECT DETECTION

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

AJEYA DEY (34200115003)

AMLAN BOSE (34200115004)

ANAMITRA CHOWDHURY (34200115005)

ORIJITA ADHIKARY (34200115032)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FUTURE INSTITUTE OF TECHNOLOGY

2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FUTURE INSTITUTE OF TECHNOLOGY

KOLKATA-700154

2019

CERTIFICATE

This is to certify that the project work entitled **"Real Time Object Detection"** submitted by **Ajeya Dey, Amlan Bose, Anamitra Chowdhury** and **Orijita Adhikary** in partial fulfilment for the award of the degree of Bachelor of Technology in **Computer Science and Engineering** at FUTURE INSTITUTE OF TECHNOLOGY is done under my guidance and supervision and is a bonafide work done by them.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other Institute/University.

Guide/Supervisor

_____

Mr. Prasanta Majumder

Department of Computer Science and Engineering

Future Institute of Technology

Date: _____

_____

Head of the Department
Computer Science and Engineering
Future Institute of Technology

ACKNOWLEDGEMENT

In performing our assignment, we have taken help and guideline of some respected persons, who deserve our greatest gratitude. The completion of this assignment gives us much pleasure. We would like to show our gratitude to Mr. Prasanta Majumder Sir for giving us a good guideline for assignment throughout numerous consultations. We would also like to thank our HOD sir Mr. Abhijit Saha and the entire faculty members.We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in writing this assignment.

Thanking You

DATE:

PLACE:

_____

AJEYA DEY (34200115003)

_____

AMLAN BOSE (34200115004)

_____

ANAMITRA CHOWDHURY (34200115005)

_____

ORIJITA ADHIKARY (34200115032)

CONTENTS

List Of Figures And Table

ABSTRACT

Efficient and accurate object detection has been an important topic in the advancement of computer vision systems. With the advent of deep learning techniques, the accuracy for object detection has increased drastically. The project aims to incorporate “You Only Look Once(YOLO)” algorithm for object detection with the goal of achieving high accuracy with a real-time performance. A major challenge in many of the object detection systems is the dependency on other computer vision techniques for helping the deep learning based approach, which leads to slow and non-optimal performance. In this project, we use a completely deep learning and neural network based approach. The resulting system is fast and accurate, thus aiding those applications which require object detection.

CHAPTER 1:

INTRODUCTION

The detection of stationary or moving targets, and tracking them on a real-time video streams is a very important and challenging task in order to protect fields from enemies. The enemies can be a human, an animal or even an object. The field is secures by drones or stationary sticks which include detection and tracking system. Video surveillance is a very popular research topic in computer vision applications that continuously tries to detect, recognize and track the targets. Object detection compromises located objects in the frame of a video sequence. Every tracking method requires a detection method in every single frame. Object tracking is the process of following one or more objects that found on detection process using a camera. Background subtraction is the most common detection method used from simple object trackers. It is based on comparing two successive frames. Mean shift method and haar cascade classifier follows this. OpenCV libraries provide us these methods and have some RGB color detection algorithms. Object tracking process can be simply defined in three steps with an order and these steps are controlled by the computer autonomously. 1. Detection of interested object in video stream. 2. Tracking target object in every single frame. 3. Analyzing the tracking process to recognize the behavior. Human brain can easily understand the visual frames which seen by eyes, but computers cannot do it without any special methodology. If we compare human and computer, camera module is the eye of the computer and the computer vision is the way that how brain understand the image. Computer vision is a field that includes some methods, processing and analyzing techniques for understanding images. Also known as image analysis, scene analysis and image understanding.

1.1. OBJECTIVE

Many problems in computer vision were saturating on their accuracy before a decade. However, with the rise of deep learning techniques, the accuracy of these problems drastically improved. One of the major problem was that of image classification, which is defined as predicting the class of the image. A slightly complicated problem is that of image localization, where the image contains a single object and the system should predict the class of the location of the object in the image (a bounding box around the object). The more complicated problem (this project), of object detection involves both classification and localization. In this case, the input to the system will be a image, and the output will be a bounding box corresponding to all the objects in the image, along with the class of object in each box. An overview of all these problems is depicted in the below figure.

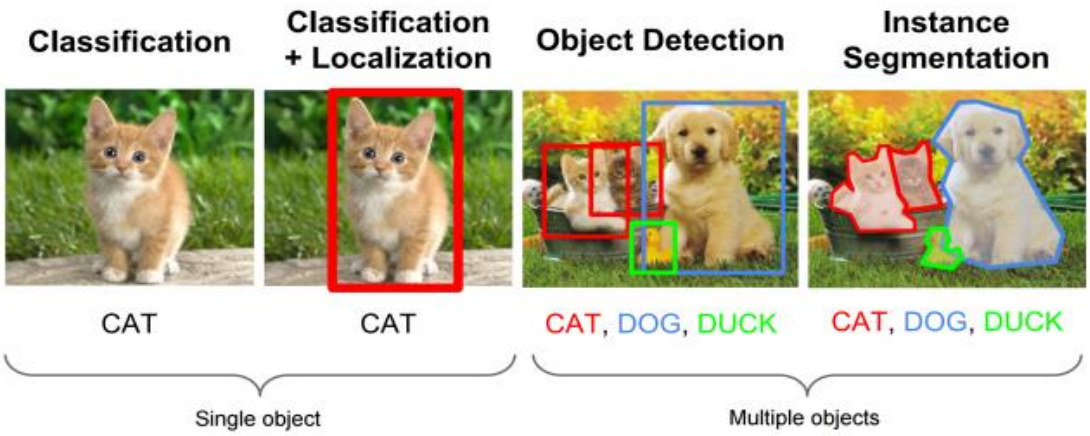

Fig.1.1 Computer Vision Tasks

1.2. OBJECT LOCALIZATION

An image classification or image recognition model simply detect the probability of an object in an image. In contrast to this, object localization refers to identifying the location of an object in the image. An object localization algorithm will output the coordinates of the location of an object with respect to the image. In computer vision, the most popular way to localize an object in an image is to represent its location with the help of bounding boxes. Fig. 1.2 shows an example of a bounding box.

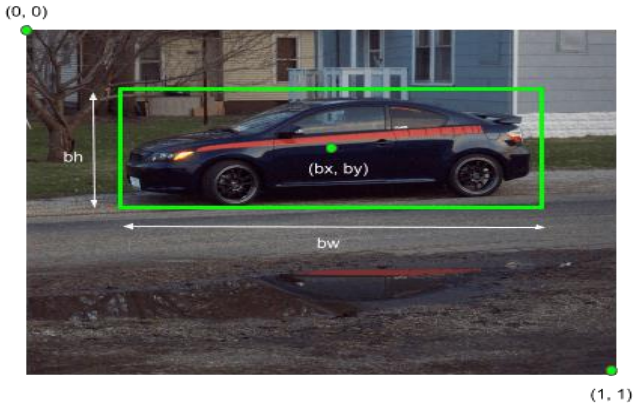

Fig. 1.2 An example of a bounding box

A bounding box can be initialized using the following parameters:

- bx, by : coordinates of the center of the bounding box
- bw : width of the bounding box w.r.t the image width
- bh : height of the bounding box w.r.t the image height

1.3. DETECTION AND TRACKING METHODS

1.3.1. Absolute Difference Method

This method can be used for detecting moving objects with respect to the background. It also called as "background subtraction". The idea behind this method is, taking two different frames and subtract each other. As a result, stationary objects will be filtered so just the moving objects will be detected.

Approach steps: 1-Take the background frame at time t.
2-Take the difference between input frame and background frame.
3-Apply the threshold to get the foreground mask.
Background subtraction equation is:

$$B(x, y, t) = I(x, y, t-1)$$

$$\downarrow$$

$$\left| I(x, y, t) - I(x, y, t-1) \right| > Th\ 5$$

Where; $B(x, y, t) \rightarrow$ Background at time t $I(x, y, t) \rightarrow$ Image at time t

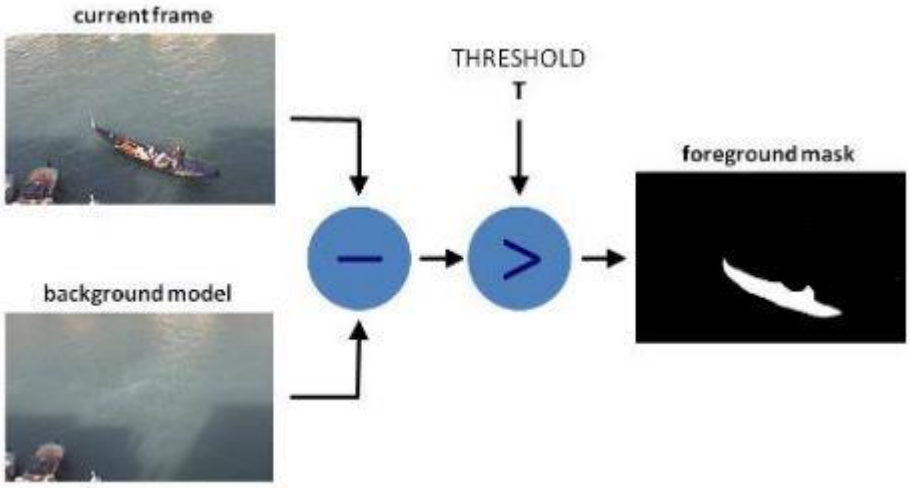

Fig. 1.3.1 Absolute difference method

This method is easy to implement. For that reason, it is widely used in most of simple object detection and tracking application. It also allows to continuous tracking. In comparison, it works slow and have low accuracy. Thus, it does not preferred in complex projects.

1.3.2. Mean Shift Analysis

Mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm.
This method is widely used on computer vision and image processing applications to detect objects. The mean shift procedure was originally presented in 1975 by Fukunaga and Hostetler. The idea behind this procedure is searching specified features(not only colors) or shapes, after that it decide the object location by analyzing the density functions of specified features. The density locations are found by using Kernel Density estimation technique. KDE is a non-parametric way to estimate the probability density function of a random

variable. Kernel density estimation is a fundamental data smoothing problem where inferences about the population are made, based on a finite data sample. It is also termed the Parzen– Rosenblatt window method, after Emanuel Parzen and Murray Rosenblatt, who are usually credited with independently creating it in its current form.

Let (x1, x2, …, xn) be an independent and identically distributed sample drawn from some distribution with an unknown density f. We are interested in estimating the shape of this function f. Its kernel density estimator is

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big),$$

Where; x = (x1, x2, …, xd)T, xi = (xi1, xi2, …, xid)T, i = 1, 2, …, n are d-vectors;

H is the bandwidth (or smoothing) d×d matrix which is symmetric and positive definite;

K is the kernel function which is a symmetric multivariate density;

h > 0 is a smoothing parameter called the bandwidth

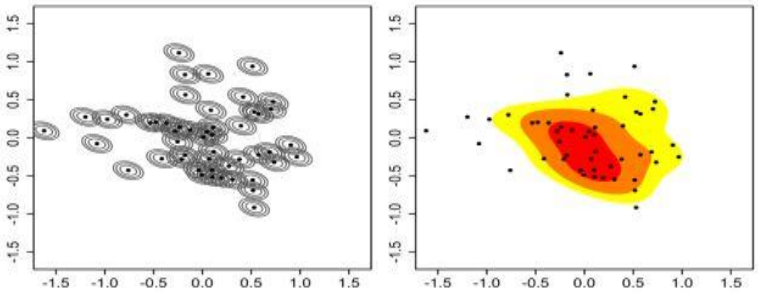

Fig. 1.3.2(a) Kernel Density & Cluster Analysis Techniques

After applying Kernel density estimation techniques more than one sets of dots. First, these dots do not have meanings. Using cluster analysis, we define these dots as corresponding meanings. Cluster analysis works as grouping similar set of objects -or colors-. Cluster analysis also allows us to filtering unrelated objects.

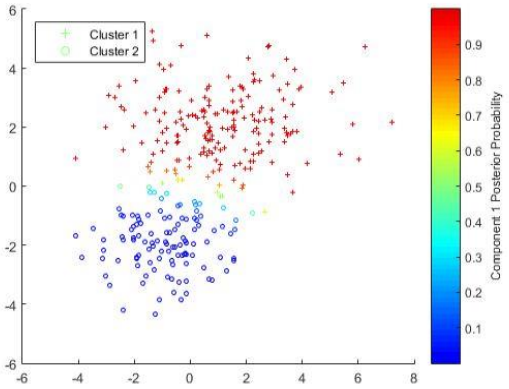

Fig. 1.3.2(b) Centroid based clustering method

Centroid based clustering method allows us to find the center of the cluster. In this method, clusters are represented by a central vector, but this vector may not be a member of the data set. Finding the center of the cluster and representing this as coordinates, helps us a lot in our project. For tracking any detected object, finding the center of cluster and while it shifts,

rotating the camera to the same direction is the easiest way in theory. With this algorithm, it is hard to miss the object in the system limitations. Mean shift method is computationally less expensive than other methods. But it is ineffective if there is a heavy occlusion.

1.3.3. Haar Cascade Classifier

Conventional object detection and tracking methods gives accurate results but they are very slow. In order to fast reaction required applications cannot use these methods. To solve this problem, Paul Viola and Micheal Jones developed haar cascade classifiers. It is also known as Viola and Jones object detection framework. The idea is to introducing the objects to the computer before detection and tracking. During the detection process, computer tries to find the introduced object, with an appropriate algorithm.

Positive image: the image that contains the desired object.

Negative object: the image that does not contain the desired object.

In order to machine learning process, positive and negative images are introduced to the computer. Then computer recognize the desired object, to using features below. First step in this process, taking the positive images and scanning it with the adjusted feature frames (shown below). Second step is to count the pixels in black and white area separately. Lastly controlling the dark and light sections and have some target values.

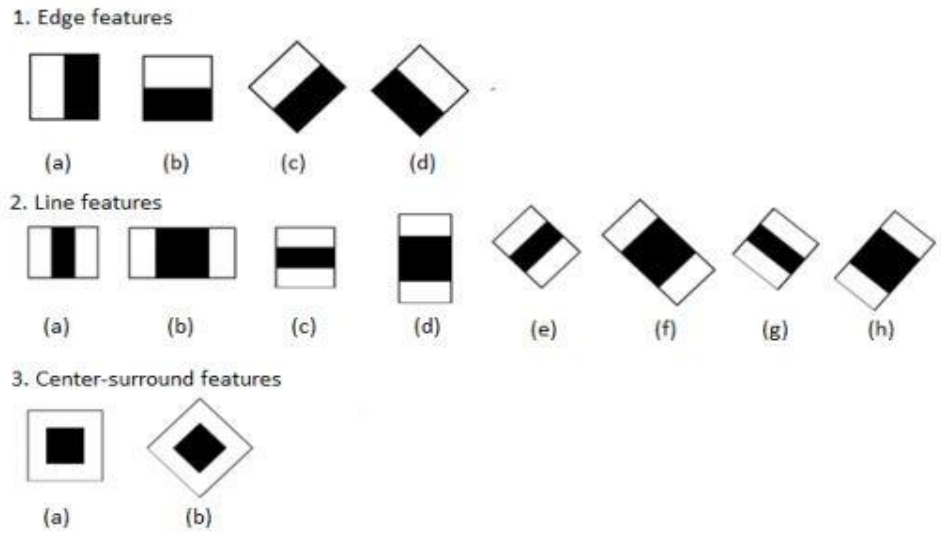

Fig. 1.3.3 Types of Feature

These frames are called as “feature” or “weak classifier”. They called weak classifier because individually they have no meanings. Most of these weak classifiers found in one object multiple times. If computer observe most of these, it means there is desired object located.

Haar cascade classifier is capable of detecting eye, nose, head, mouth, hair, etc. in face detection application. While detection weak classifier sizes are adjusted dynamically.

Negative images are also scanned by the computer with the same algorithm. The result of it, computer knows what to do not want to detect, and this saves a lot of time.

Haar cascade classifier's speed secret can be easily explained with one word, integration. Instead of adding all the pixel variables separately, taking integral of it is much easier and faster.

Fortunately, the computer vision software which named as OpenCV has a library of haar cascade classifier. This library is designed with state-of-art algorithm to be user friendly, time saver, and accurate.

1.4. Object Detection Vs Classification

People often confuse image classification and object detection scenarios. In general, if we want to classify an image into a certain category, we use image classification. On the other hand, if we aim to identify the location of objects in an image, and, for example, count the number of instances of an object, we can use object detection.

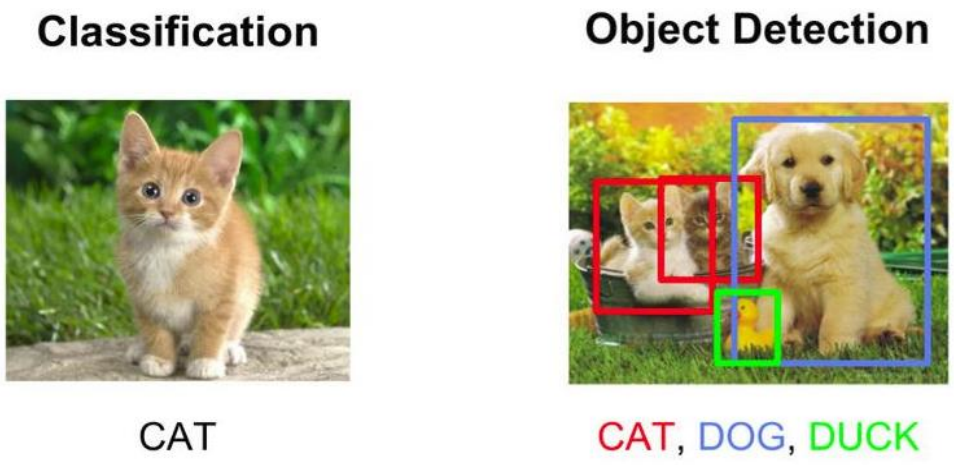

Fig. 1.4 Image classification and object detection scenarios

There is, however, some overlap between these two scenarios. If we want to classify an image into a certain category, it could happen that the object or the characteristics that are required to perform categorisation are too small with respect to the full image. In that case, we would achieve better performance with object detection instead of image classification even if we are not interested in the exact location or counts of the object.

Imagine we need to check circuit boards and classify them as either defect or correct. While it is essentially a classification problem, the defects might be too small to be noticeable with an image classification model. Constructing an object detection dataset will cost more time, yet it will result most likely in a better model.

1.5. General Object Detection Framework

Typically, there are three steps in an object detection framework.

1. First, a model or algorithm is used to generate regions of interest or region proposals.

These region proposals are a large set of bounding boxes spanning the full image (that is, an object localization component).

2. In the second step, visual features are extracted for each of the bounding boxes, they are evaluated and it is determined whether and which objects are present in the proposals based on visual features (i.e. an object classification component).

3. In the final post-processing step, overlapping boxes are combined into a single bounding box (that is, non maximum suppression).

1.6. EVALUATION METRIC

The most common evaluation metric that is used in object recognition tasks is 'mAP', which stands for **'mean average precision'**. It is a number from 0 to 100 and higher values are typically better, but it's value is different from the accuracy metric in classification.

Each bounding box will have a score associated (likelihood of the box containing an object). Based on the predictions a precision-recall curve (PR curve) is computed for each class by varying the score threshold. The average precision (AP) is the area under the PR curve. First the AP is computed for each class, and then averaged over the different classes. The end result is the mAP.

Note that a detection is a true positive if it has an **'intersection over union'** (IoU or overlap) with the ground-truth box greater than some threshold (usually 0.5). Instead of using mAP we typically use mAP@0.5 or mAP@0.25 to refer to the IoU that was used.

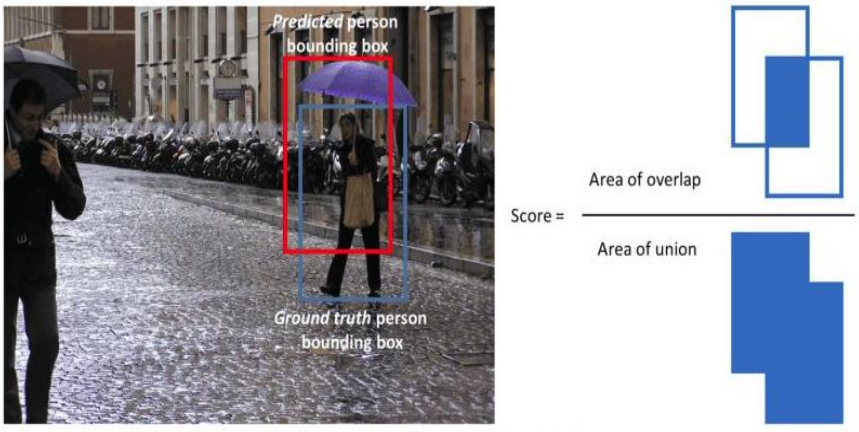

Fig. 1.6 A visualisation of the definition of IoU

1.7. ALGORITHMS FOR OBJECT DETECTION

The difference between object detection algorithms and classification algorithms is that in

detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image. Also, we might not necessarily draw just one bounding box in an object detection case, there could be many bounding boxes representing different objects of interest within the image and we would not know how many beforehand.

1.7.1. SSD (Single Shot Multibox Detector)

Overview

The SSD architecture was published in 2016 by researchers from Google. It presents an object detection model using a single deep neural network combining regional proposals and feature extraction.

A set of default boxes over different aspect ratios and scales is used and applied to the feature maps. As these feature maps are computed by passing an image through an image classification network, the feature extraction for the bounding boxes can be extracted in a single step. Scores are generated for each object category in every of the default bounding boxes. In order to better fit the ground truth boxes adjustment offsets are calculated for each box.

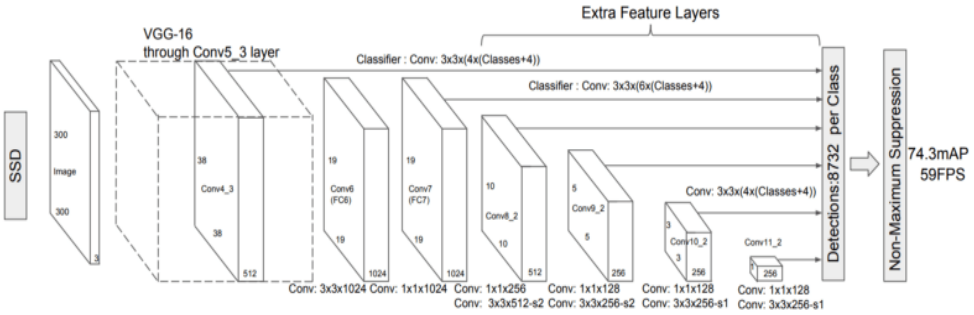

Fig. 1.7.1 The SSD network leveraging feature maps from VGG-16

Different feature maps in the convolutional network correspond with different receptive fields and are used to naturally handle objects at different scales. As all the computation is encapsulated in a single network and fairly high computational speeds are achieved (for example, for 300 × 300 input 59 FPS).

Usage

For the usage we will investigate the different sample configuration files for SSD. Several parameters are important when leveraging the SSD architecture and we will go over them one by one.

First, different classification networks have different strengths and weaknesses. The Inceptionv3 network for example is trained to detect objects well at different scales, whereas the ResNet architecture achieves very high accuracy overall. Mobilenet on the other is a network that was trained to minimise the required computational resources. The performance of the feature extraction network on ImageNet, the number of parameters and the original dataset it was trained on are a good proxy for the performance/speed tradeoff. The feature extractor is defined in the 'feature_extractor' section.

A second obvious set of parameters are the settings for the default boxes and aspect ratios. Depending on the type of problem, it is worthwhile to analyse the various aspect ratios and scales of the bounding boxes of the labeled data. Setting the aspect ratios and scales will ensure that the network does not do unnecessary calculations. We can tweak these in the 'ssd_anchor_generator' section. Note that adding more scales and aspect ratios will lead to better performance, but typically with diminishing returns.

Thirdly, when training the model it is important to set the image size and data augmentation options in the 'data_augmentation_options' and 'image_resizer' sections. A larger image size will perform better as small object are often hard to detect, but it will have a significant computational cost. Data augmentation is especially important in the context of SSD in order to be able to detect objects at different scales (even at scales which might not be present in the training data).

Finally, tweaking the 'train_config', setting the learning rates and batch sizes is important to reduce overfitting, and will highly depend on the size of the dataset you have.

1.7.2. Faster R-CNN

Overview

Faster R-CNN was developed by researchers at Microsoft. It is based on R-CNN which used a multi-phased approach to object detection. R-CNN used Selective search to determine region proposals, pushed these through a classification network and then used an SVM to classify the different regions.

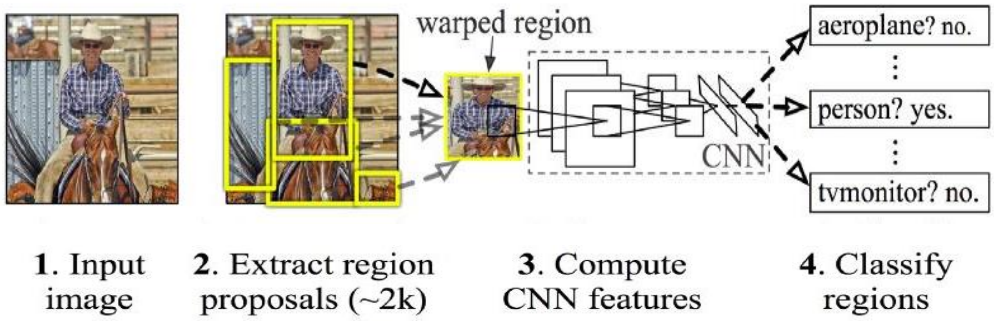

Fig. 1.7.2(a) An overview of the R-CNN architecture

Faster R-CNN, similar to SSD, is an end-to-end approach. Instead of using default bounding boxes, Faster R-CNN has a Region Proposal Network (RPN) to generate a fixed set of regions. The RPN uses the convolutional features from the the image classification network, enabling nearly cost-free region proposals. The RPN is implemented as a fully convolutional network that predicts object bounds and objectness scores at each position.

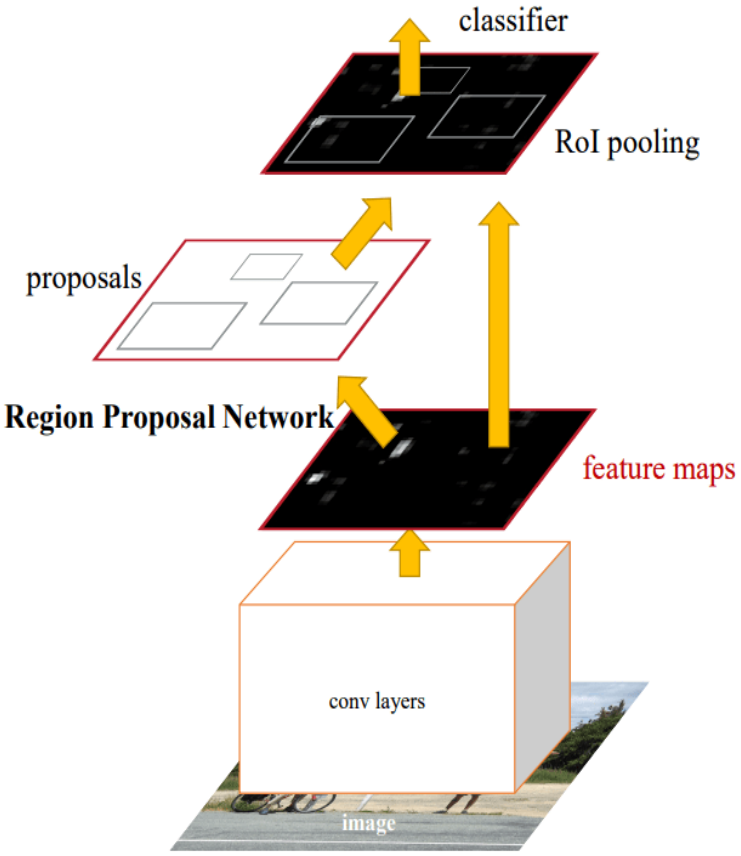

Fig. 1.7.2(b) An overview of Faster-RCNN

Note that the RPN has a similar setup as the the SSD network (i.e. it does not predict bounding boxes out of thin air). The RPN network works with sliding windows across the feature maps. At each sliding-window location or anchor, a set of proposals are computed with various scales and aspect ratios. Similar to SSD, the outcome of the RPN are 'adjusted' bounding boxes, based on the anchors.

The different components are combined in a single setup and are trained either end-to-end or in multiple phases (to improve stability). Another interpretation of the RPN is that it guides the networks 'attention' to interesting regions.

Usage

Most of the usage details of Faster R-CNN are similar as the ones for SSD. In terms of raw mAP, Faster R-CNN typically outperforms SSD, but it requires significantly more computational power.

An important section for the Fast-RCNN detector, is the 'first_stage_anchor_generator' which defines the anchors generated by the RPN. The strides in this section defines the steps of the sliding window. Note that especially when attempting to detect small objects (if the stride is too large, we might miss them).

Although no extensive data augmentation was used by the authors of the Faster-RCNN paper, it is still advised to use it when working with smaller datasets.

CHAPTER 2:

YOLO: REAL TIME OBJECT DETECTION

YOLO (You Only Look Once), is a network for object detection. The object detection task consists in determining the location on the image where certain objects are present, as well as classifying those objects. Previous methods for this, like R-CNN and its variations, used a pipeline to perform this task in multiple steps. This can be slow to run and also hard to optimize, because each individual component must be trained separately. YOLO, does it all with a single neural network.

2.1. YOLO ALGORITHM

There are a few different algorithms for object detection and they can be split into two groups:

1. Algorithms based on classification – they work in two stages. In the first step, we're selecting from the image interesting regions. Then we're classifying those regions using convolutional neural networks. This solution could be very slow because we have to run prediction for every selected region. Most known example of this type of algorithms is the Region-based convolutional neural network (RCNN) and their cousins Fast-RCNN and Faster-RCNN.
2. Algorithms based on regression – instead of selecting interesting parts of an image, we're predicting classes and bounding boxes for the whole image **in one run of the algorithm**. Most known example of this type of algorithms is **YOLO (You only look once)** commonly used for real-time object detection.

Before we go into YOLOs details we have to know what we are going to predict. Our task is to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

1. center of a bounding box ($\mathbf{b_x}$ $\mathbf{b_y}$)
2. width ($\mathbf{b_w}$)
3. height ($\mathbf{b_h}$)
4. value **c** is corresponding to a class of an object (f.e. car, traffic lights,…).

We've got also one more predicted value p_c which is a probability that there is an object in the bounding box, I will explain in a moment why do we need this.

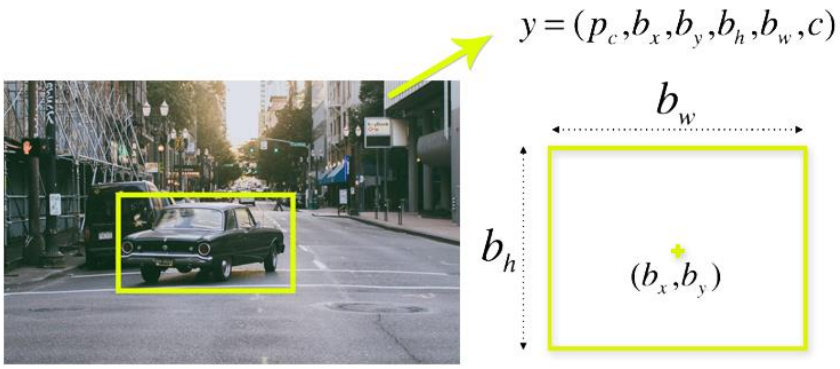

Fig. 2.1(a) Predicting a class of an object and the bounding box specifying object location

Like I said before with YOLO algorithm we're not searching for interested regions on our image that could contain some object. Instead of that we are splitting our image into cells, typically its 19×19 grid. Each cell will be responsible for predicting 5 bounding boxes (in case there's more than one object in this cell). This will give us 1805 bounding boxes for an image and that's a really big number!

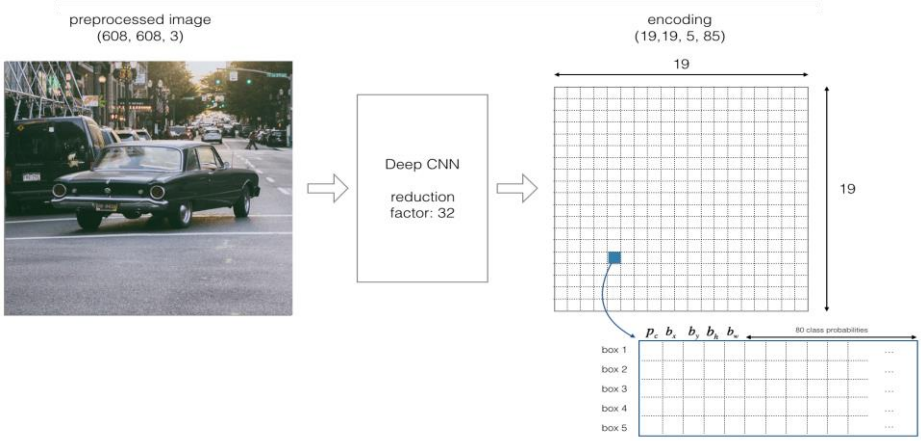

Fig. 2.1(b) How YOLO works

Majority of those cells and boxes won't have an object inside and this is the reason why we need to predict p_c. In the next step, we're removing boxes with low object probability and bounding boxes with the highest shared area in the process called **non-max suppression**.

Fig. 2.1(c) Before and after non-max suppression

2.2. THE PREDICTIONS VECTOR

The first step to understanding YOLO is how it encodes its output. The input image is divided into an S x S grid of cells. For each object that is present on the image, one grid cell is said to be "responsible" for predicting it. That is the cell where the center of the object falls into.
Each grid cell predicts B bounding boxes as well as C class probabilities. The bounding box prediction has 5 components: (x, y, w, h, confidence). The (x, y)coordinates represent the center of the box, relative to the grid cell location (remember that, if the center of the box does not fall inside the grid cell, than this cell is not responsible for it). These coordinates are normalized to fall between 0 and 1. The (w, h) box dimensions are also normalized to [0, 1], relative to the image size. Let's look at an example:

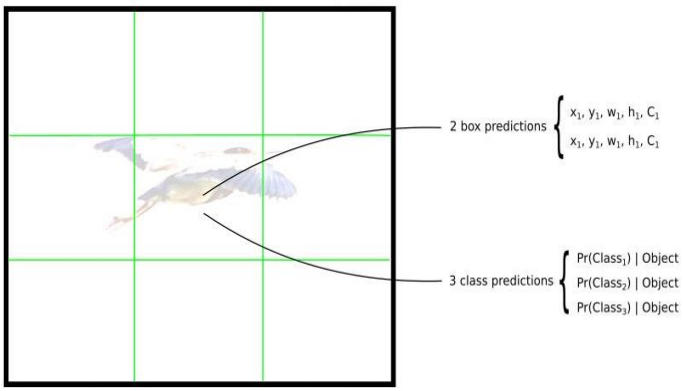

Fig. 2.2 <u>Each grid cell makes B bounding box predictions and C class predictions (S=3, B=2 and C=3 in this example)</u>

2.3. THE LOSS FUNCTION

There is a lot to say about the loss function, so let's do it by parts. It starts like this:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

<u>YOLO Loss Function—Part 1</u>

This equation computes the loss related to the predicted bounding box position **(x,y)**. Don't worry about **λ** for now, just consider it a given constant. The function computes a sum over each bounding box predictor **(j = 0.. B)** of each grid cell **(i = 0 .. S^2)**. **$\mathbb{1}_{obj}$** is defined as follows:

- 1, If an object is present in grid cell i and the jth bounding box predictor is "responsible" for that prediction
- 0, otherwise

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth.

The other terms in the equation should be easy to understand: **(x, y)** are the predicted bounding box position and **(x̂, ŷ)** hat are the actual position from the training data.

Let's move on to the second part:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

YOLO Loss Function—Part 2

Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

Moving on to the third part:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

YOLO Loss Function—Part 3

Here we compute the loss associated with the confidence score for each bounding box predictor. **C** is the confidence score and **Ĉ** is the intersection over union of the predicted bounding box with the ground truth.**𝟙 obj** is equal to one when there is an object in the cell, and 0 otherwise. **𝟙 noobj** is the opposite.
The **λ** parameters that appear here and also in the first part are used to differently weight parts of the loss functions. This is necessary to increase model stability. The highest penalty is for coordinate predictions (**λ coord** = 5) and the lowest for confidence predictions when no object is present (**λ noobj** = 0.5).
The last part of the loss function is the classification loss:

$$\sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

YOLO Loss Function—Part 4

It looks similar to a normal sum-squared error for classification, except for the **𝟙 obj** term. This term is used because so we don't penalize classification error when no object is present on the cell (hence the conditional class probability discussed earlier).

2.4. COMPARISON TO OTHER DETECTOR

YOLOv3 is extremely fast and accurate. In Map measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, one can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required!

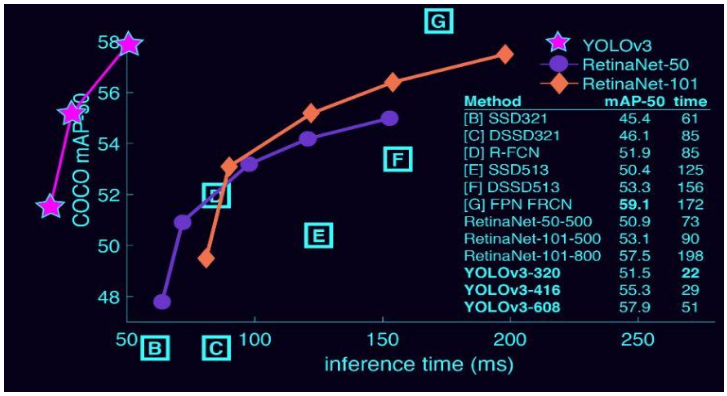

Fig. 2.4 COCO vs Inference time graph

2.5. PERFORMANCE ON COCO DATASET

We'll be using YOLOv3 in this PROJECT, in particular, YOLO trained on the COCO dataset.

The COCO dataset consists of 80 labels, including, but not limited to:

- People
- Bicycles
- Cars and trucks
- Airplanes
- Stop signs and fire hydrants
- Animals, including cats, dogs, birds, horses, cows, and sheep, to name a few
- Kitchen and dining objects, such as wine glasses, cups, forks, knives, spoons, etc.

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Fig. 2.5 Performance of the COCO Model

2.6. HOW YOLO WORKS

All prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

YOLO uses a totally different approach. YOLO applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

2.7. YOLO MODEL COMPARISON

- yolo.cfg is based on the extraction network. It processes images at 45 fps, here are weight files for yolo.cfg trained on 2007 train/val+ 2012 train/val, and trained on all 2007 and 2012 data.
- yolo-small.cfg has smaller fully connected layers so it uses far less memory. It processes images at 50 fps, here are weight files for yolo-small.cfg trained on 2007 train/val+ 2012 train/val.
- yolo-tiny.cfg is much smaller and based on the Darknet reference network. It processes images at 155 fps, here are weight files for yolo-tiny.cfg trained on 2007 train/val+ 2012 train/val.

CHAPTER 3:

REQUIRMENT

In order to run the object detection we need to have some libraries installed.

3.1. DARKNET

We used an open source neural network framework called **Darknet**. Darknet was written in C language and CUDA technology, what makes it really fast and allows you to make computations on a GPU, which is essential for real-time predictions.

3.2. OPEN CV

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

3.3. PRE TRAINED WEIGHTS

Darknet prints out the objects it detected, its confidence, and how long it took to find them. Since we are using Darknet on the CPU it takes around 6-12 seconds per image. If we use the GPU version it would be much faster. We downloaded the weight from **http://pjreddie.com/media/files/yolo.weights**.

A Smaller Model

The original YOLO model uses a lot of GPU memory. If we have a smaller graphics card we can try using the smaller version of the YOLO model, yolo-small.cfg. We downloaded the model from **http://pjreddie.com/media/files/yolo-small.weights**
The small version of YOLO only uses 1.1 GB of GPU memory so it should be suitable for many smaller graphics cards.

A Tiny Model

The yolo-tiny.cfg is based on the Darknet reference network. We should already have the config file in the cfg/ subdirectory.
The pre trained weights here http://pjreddie.com/media/files/yolo-tiny.weights
The tiny version of YOLO only uses 611 MB of GPU memory and it runs at more than 150 fps.

CHAPTER 4:

PROPOSED WORK

4.1. CODE EXPLANATION

The model we used was trained on COCO dataset.

First we import the necessary packages

```
import cv2
import argparse
import numpy as np
```

The first step when using argparse is to create a parser object and tell it what arguments to expect. The parser can then be used to process the command line arguments when our program runs.

The parser class is argument parser. The constructor takes several arguments to set up the description used in the help text for the program and other global behaviors or settings.

Then we load the necessary arguments which can trigger different actions

```
ap.add_argument('-c', '--config',
help = 'path to yolo configfile',default = 'yolov3.cfg')
ap.add_argument('-w', '--weights',
help = 'path to yolo pre-trained weights', default = 'yolov3.weights')
ap.add_argument('-cl', '--classes',
help = 'path to text file containing class names', default = 'yolov3.txt')
args = ap.parse_args()
```

Generally in a sequential CNN network there will be only one output layer at the end. In the YOLO v3 architecture we are using there are multiple output layers giving out predictions. get_output_layers() function gives the names of the output layers. An output layer is not connected to any next layer.

```
def getOutputsNames(net):
layersNames = net.getLayerNames()
return [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```

Now we will draw a rectangle over the object which we predict in real time and we will also define the class in which it falls

```
defdraw_pred(img, class_id, confidence, x, y, x_plus_w, y_plus_h):

label = str(classes[class_id])

color = COLORS[class_id]

cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)

cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

Since it is real time detection we will open a new window to show the stream

```
window_title= "OBJECT DETECTOR"
cv2.namedWindow(window_title, cv2.WINDOW_NORMAL)
```

Now we will load all the classes from the model,this particular model is trained on COCO dataset (common objects in context) from Microsoft. It is capable of detecting 80 common objects.

```
classes = None
with open(args.classes, 'r') as f:
classes = [line.strip() for line in f.readlines()]
print(classes)
```

Now we have to define colour for all the classes and load the weights

```
COLORS = np.random.uniform(0, 255, size=(len(classes), 3))
net = cv2.dnn.readNet(args.weights,args.config)
```

Now we have to define the video capture for the default camera and set the resolution and also check for the image we get and try to predict the output

```
cap = cv2.VideoCapture(0)


while cv2.waitKey(1) < 0 or False:

hasframe, image = cap.read()
image=cv2.resize(image, (1280, 720))

blob = cv2.dnn.blobFromImage(image, 1.0/255.0, (416,416), [0,0,0], True, crop=False)
    Width = image.shape[1]
    Height = image.shape[0]
net.setInput(blob)

outs = net.forward(getOutputsNames(net))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4
```

```
for out in outs:
        #print(out.shape)
for detection in out:

        #each detection  has the form like this [center_xcenter_y width height obj_score class_1_score class_2_score ..]
scores = detection[5:]#classes scores starts from index 5
class_id = np.argmax(scores)
confidence = scores[class_id]
if confidence > 0.5:
center_x = int(detection[0] * Width)
center_y = int(detection[1] * Height)
                w = int(detection[2] * Width)
                h = int(detection[3] * Height)
                x = center_x - w / 2
                y = center_y - h / 2
class_ids.append(class_id)
confidences.append(float(confidence))
boxes.append([x, y, w, h])
```

Finally we apply non max suppression and put the efficiency value and predict it

```
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

for i in indices:
i = i[0]
box = boxes[i]
        x = box[0]
        y = box[1]
        w = box[2]
        h = box[3]
draw_pred(image, class_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))


t, _ = net.getPerfProfile()
label = 'Inference time: %.2f ms' % (t * 1000.0 / cv2.getTickFrequency())
cv2.putText(image, label, (0, 15), cv2.FONT_HERSHEY_SIMPLEX, .6, (255, 0, 0))

cv2.imshow(window_title, image)
```

4.2. EXPERIMENTAL RESULTS

In the testing section we have used some objects like bag, cup, brush, cell phone, vase, umbrella, book, clock, chair, bed, etc. to find the predicted results. Few of those snapshots are provided below:

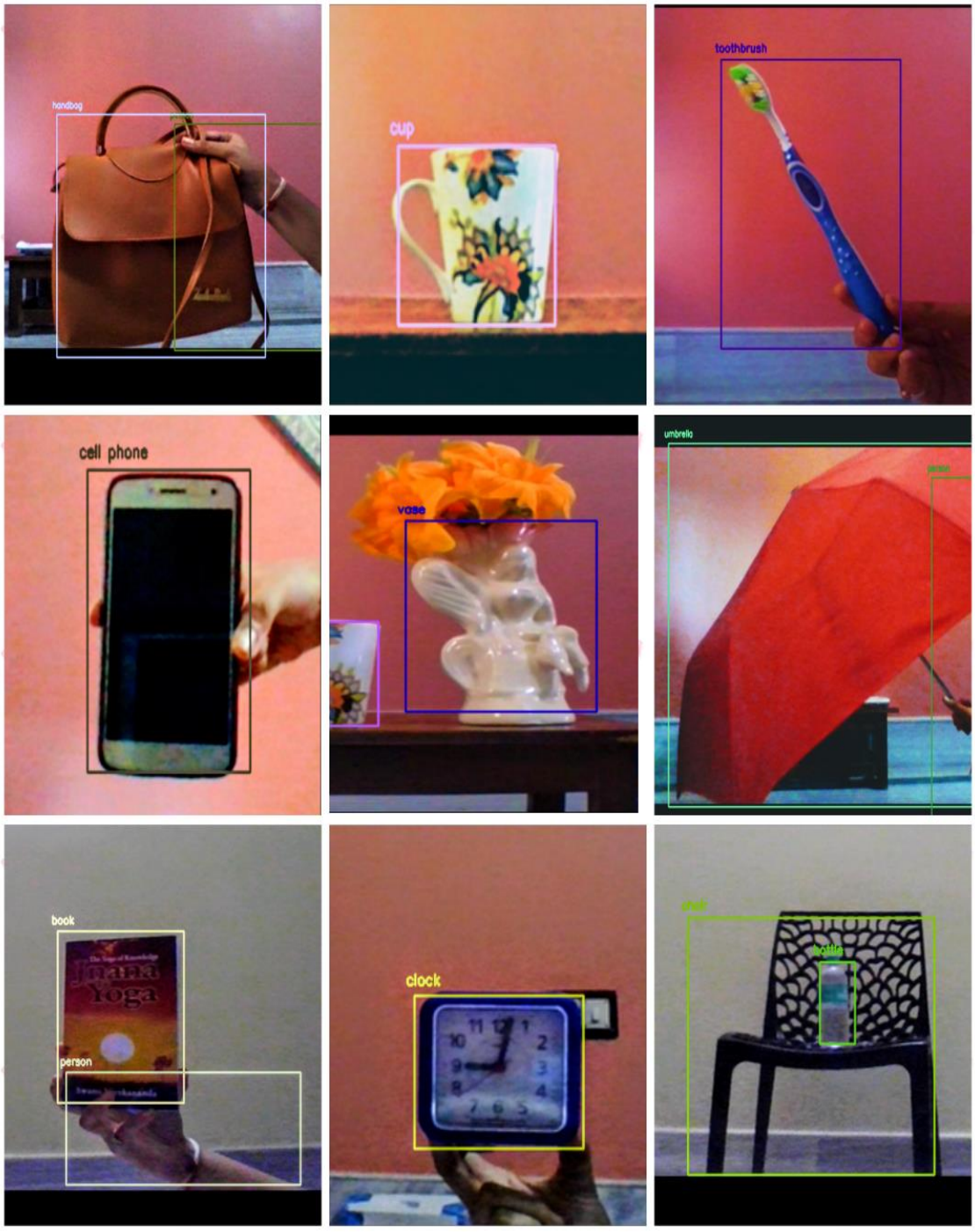

Fig. 4.2 Objects detected successfully

CHAPTER 5:

CONCLUSION

An accurate and efficient object detection system has been developed which achieves comparable metrics with the existing YOLO system. This project uses recent techniques in the field of neural network and deep learning. Custom dataset was created using label image and the evaluation was consistent. This can be used in real-time applications which require object detection for pre-processing in their pipeline. An important scope would be to train the system on a video sequence for usage in tracking applications. Addition of a temporally consistent network would enable smooth detection and more optimal than per-frame detection.

CHAPTER 6:

FUTURE SCOPE

In future our project will improve and may help in the following ways:-

- Recognize the specific instance (e.g., to identify the subject's face)
- Track the object over an image sequence (e.g., to track the face in a video)
- Extracting information (e.g., detecting a piece of text from a book)
- Human-computer interaction (HCI)
- Security (e.g., recognition, tracking)
- Retrieval (e.g., search engines, photo management)

CHAPTER 7:

BIBLIOGRAPHY

- https://pjreddie.com/media/files/papers/yolo.pdf
- YOLO — You only look once, real time object detection explained
  https://towardsdatascience.com/yolo-you-only-look-once-real-time
- What's new in YOLO v3? – Towards Data Science
  https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
- Understanding YOLO – Hacker Noon
  https://hackernoon.com/understanding-yolo-f5a74bbc7967
- You Only Look Once: Unified, Real-Time Object Detection
  https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/
- A Practical Guide to Object Detection using the Popular YOLO
  https://www.analyticsvidhya.com/blog/2018/12/practical-guide

THANK YOU