1. INTRODUCTION

Interest in the potential of digital images has increased enormously over the last few years with the rapid growth of imaging in World Wide Web. Users in almost every field are interested in opportunities to access and manipulate remotely-stored images. However, the process of locating a desired image from a large and varied collection can be cumbersome task.

Earlier, the method called Text Based Image Retrieval(TBIR) was used for the image retrieval process.In this method, images are retrieved on the basis of keywords and tags associated with it and those tags were to be stored in the database with the respective image manually. Thus it is not possible to assign keyword to huge amount of images. And to overcome this drawback, Content Based Image Retrieval(CBIR) was introduced.It is a technique for retrieving a specific image from database based on the content of the image.Content is nothing but the features like shape,texture,colour etc.

1.1 BACKGROUND

Growth of Digital Imaging

The unparalleled growth in number and availability of images in the twentieth century can be attributed to the use of images in all walks of life. Images are being used in all sectors , especially journalism , medicine,education and entertainment. Technological inventions are the major cause in the sudden increase of image data. Images,pictures were being used earlier as well but technological advancements allowed their use in the digital form as well. In the early 1990's , the creation of the World Wide Web gave a sudden boost to the exploitation of digital images.

1.2 Need of Image Data Management

With the sudden increase in the number of digital images being produced daily. A mechanism was required for proper management of this vast collection of images. The process of digitisation did not itself make image management an easy task.
Some form of indexing and cataloging was needed to make the storage and retrieval process - speedy and accurate. One of the main problems of image data management was to locate a desired image in a large collection . Journalists,engineers, designers all needed some kind of access by image content.

1.3 What is CBIR ?

Content Based Image Retrieval is a technique of retrieving a desired image from a vast collection of images based on the content of the image. The retrieval of images is based on features - colour, shape and texture.
Before the introduction of CBIR , Text Based Image Retrieval was used. Each image in the database was associated with a keyword or tag manually. Since it was a manual process it was very time consuming. Moreover the limitation with Text Based Image Retrieval was that it required the user to annotate the image with the text (metadata) that is considered relevant. However different people would perceive the same image differently , this would lead to mismatches in the retrieval process later.
To overcome these limitations,Content Based Retrieval was proposed in the early 1990’s. It aimed at extracting certain features from the image rather than relying on human annotation with metadata. Hence CBIR was more accurate and faster retrieval process.

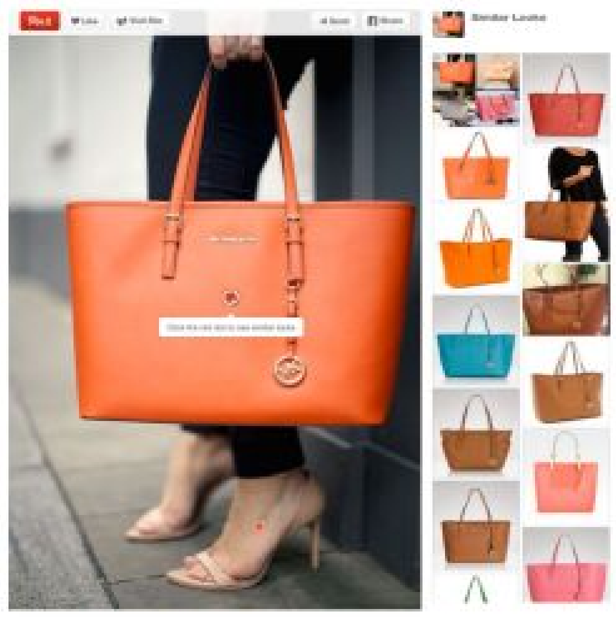


Fig. 1: Image Retrieval

1.3 OBJECT DETECTION AND ITS NEED

Object detection is a vital, yet difficult vision task. It is a basic part in numerous applications, for example, image search, image auto-annotation and scene understanding;

be that as it may it is as yet an open issue because of the intricacy of object classes and images. It is being widely used in industries to ease user, save time and to achieve parallelism. Object detection is a part of computer vision which aims at having a human like vision,which can locate and differentiate various objects such as, numbers,location,size, position etc. The common object detection method is the color-based approach, detecting objects based on their color values.

Object detection is one of the most challenging applications of the image processing. It is a branch of computer vision and artificial intelligence. The aim of this project is to identify and locate the objects in the images or videos and also naming the specific objects detected.

Object Detection comes under **Convolutional Neural Network (CNN).**

Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. In this article we will discuss the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization. See the respective tutorials on convolution and pooling for more details on those specific operations.

Architecture

A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m x m x rm x m x r image where mm is the height and width of the image and rr is the number of channels, e.g. an RGB image has r=3r=3. The convolutional layer will have kk filters (or kernels) of size n x n x qn x n x q where nn is smaller than the dimension of the image and qq can either be the same as the number of channels rr or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce kk feature maps of size m−n+1m−n+1. Each map is then subsampled typically with mean or max pooling over p x pp x p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights.

2. Motivation

Every day the digital images data are growing explosively. Photography and television technology forms the invention that have played major role in facilitating the communication and capture of image data. The digitization process cannot manage collection of images by itself easily. It requires catalogue and indexing. Images are important in electronically-mediated communication. It is difficult to locate a particular image in a varied and huge image collection by simply searching and uploading. Earlier image retrieval was based on the text based or annotation assigned to different images. But querying a wrongly annotated image by text does not give required image in result. There is a need of fast and secured technique to upload, search and retrieve these images on user demand**.**

The modern world is encircled with heavy masses of digital visual information, may it be images,videos and so on. It is a need of hour to analyse and organise these plundering ocean of visual information and for these techniques are required. Particularly,it will be more useful to analyse semantic information of the images or videos.One imperative part of image content is the objects in the image. So there is a need for object detection and image recognition techniques.

2.1 Limitations of Text Based Image Retrieval

Textual based retrieval cannot append the perceptual significant visual features like color, shape, texture. The other difficulty comes from the rich content in the images and the subjectivity of human perception which is more essential. The annotation of the image and videos completely depends on the annotation interpretation i.e. different people may perceive the same image differently.

The users may find it difficult to use text to perform a query for some portion of the content of an image or video. Text-based retrieval techniques are absolutely limited to search the metadata that is tagged to the image or video. If the text queried is not annotated with the same tag as attached with the image or video, the data is not returned. This means that if a particular piece of the image or video is interesting this must be explicit included in the metadata. If the desired object is not a main part of the image or video, sometimes it may happen that is not described in the metadata and hence cannot be a retrieve as a result from a query describing such portions of the image or video.

2.2 Content Based Image Retrieval

Many retrieval methods that accept query images as input from the user represent images as vectors in the feature space and search for images based on their features and feature representations. When the user presents a sample query image, region of interest (ROI), or pattern to the system, it performs various visual query mechanisms, such as the query-by example (QBE) paradigm, and finally outputs the relevant images. In CBIR, image content is frequently represented using image features. CBIR finds applications in internet, advertising, medicine, crime detection, entertainment, and digital libraries. High retrieval efficiency and less computational complexity are the desired characteristics of CBIR system and they are the key objectives in the design of a CBIR system.

Applications of CBIR :

Content-Based Image Retrieval has been used in several applications, such as medicine, fingerprint identification, biodiversity information systems, digital libraries, crime prevention, historical research, among others.

A. Medical Applications

The number of medical images produced by digital devices has increased more and more. For instance, a medium-sized hospital usually performs procedures that generate medical images that require hundreds or even thousands of gigabytes within a small space of time. The task of taking care of such huge amount of data is hard and time-consuming. That’s one of the reasons that has motivated research in the field of Content-Based Image Retrieval. In fact, the medical domain is frequently mentioned as one of the main areas where Content Based Image Retrieval finds its application.

B. Biodiversity Information Systems

Biologists gather many kinds of data for biodiversity studies, including spatial data, and images of living beings.
Ideally, Biodiversity Information Systems (BIS) should help researchers to enhance or complete their knowledge and understanding about species and their habitats by combining textual, image content-based, and geographical queries. An example of such a query might start by providing an image as input (e.g., a photo of a fish) and then asking the system to "Retrieve all database images containing fish whose fins are shaped like those of the fish in this photo".

C. Digital Libraries

There are several digital libraries that support services based on image content. One example is the digital museum of butterflies, aimed at building a digital collection of Taiwanese butterflies. This digital library includes a module responsible for content-based image retrieval based on color, texture, and Patterns.

D. Crime Prevention

The police use visual information to identify people or to record the scenes of crime for evidence; over the course of time, these photographic records become a valuable archive. In the UK, it is common practice to photograph everyone who is arrested and to take their fingerprints. The photograph will be filed with the main record for the person concerned, which in a manual system is a paper-based file. In a computer-based system, the photograph will be digitised and linked to the corresponding textual records. Until convicted, access to photographic information is restricted and, if the accused is acquitted, all photographs and fingerprints are deleted. If convicted, the fingerprints are passed to the National Fingerprint Bureau. Currently, there is a national initiative investigating a networked Automated Fingerprint Recognition system involving BT and over thirty regional police forces. Other uses of images in law enforcement include face recognition , DNA matching, shoe sole impressions, and surveillance systems. The Metropolitan Police Force in London is involved with a project which is setting up an international database of the images of stolen objects

E. Fashion and graphic design

Imagery is very important for graphic, fashion and industrial designers. Visualisation seems to be part of the creative process. Whilst there will be individual differences in the way designers approach their task, many use images of previous designs in the form of pictures, photographs and graphics, as well as objects and other visual information from the real world, to provide inspiration and to visualise the end product. 2-D sketches, and, increasingly, 3-D geometric models are used to present ideas to clients and other colleagues. There is also a need to represent the way garments hang and flow.

F. Publishing and advertising

Photographs and pictures are used extensively in the publishing industry, to illustrate books and articles in newspapers and magazines. Many national and regional newspaper publishers maintain their own libraries of photographs, or will use those available from the Press Association, Reuters and other agencies. The photographic collections will be indexed and filed under, usually, broad subject headings (e.g. local scenes, buildings or personalities as well as pictures covering national and international themes). Increasingly, electronic methods of storage and access are appearing, alongside developments in automated methods of newspaper production, greatly improving the speed and accuracy of the retrieval process. Advertisements and advertising campaigns rely heavily on still and moving imagery to promote the products or services. The growth of commercial stock photograph libraries, such as Getty Images and Corbis, reflects the lucrative nature of the industry.

G. Education and training

It is often difficult to identify good teaching material to illustrate key points in a lecture or self-study module. The availability of searchable collections of video clips providing examples of (say) avalanches for a lecture on mountain safety, or traffic congestion for a course on urban planning, could reduce preparation time and lead to improved teaching quality. In some cases (complex diagnostic and repair procedures) such videos might even replace a human tutor.

Reports of the application of CBIR technology to education and training have so far been sparse - though Carnegie-Mellon University's Informedia system is being trialled at a number of universities, including the Open University in the UK [van der Zwan et al, 1999]. It appears to be too early to form any definite conclusions about the system's effectiveness in practice.

2.3 Current Technologies for Image Retrieval

The last decade saw a number of commercial image management systems being developed. These systems did not have CBIR facilities. They were based on text keywords that required a human indexer to associate each image with a keyword.
Thus the Text Based Image Retrieval which was being used, was later on replaced by Content Based Image Retrieval. It is based on the content of images that include colour , shape and texture.It is efficient for low dimensions but inefficient for high dimensions as effective storage and speedy retrieval is needed and traditional data structures are not sufficient. So, Content Based Image Retrieval Using Hadoop will be efficient in storing content and the speedy, accurate retrieval of images will be possible. Three **commercial CBIR** systems are now available - IBM's QBIC, Virage's VIR Image Engine, and VisualSeek .

IBM’s Query By Image Content (QBIC) -

Engineers at IBM have developed the QBIC (Query by Image Content) system to explore content-based retrieval methods. IBM's QBIC system is probably the best-known of all image content retrieval systems. It is available commercially either in standalone form, or as part of other IBM products such as the DB2 Digital Library. It offers retrieval by any combination of colour, texture or shape - as well as by text keyword. Image queries can be formulated by selection from a palette, specifying an example query image, or sketching a desired shape on the screen. The system extracts and stores colour, shape and texture features from each image added to the database, and uses R*-tree indexes to improve search efficiency .At search time, the system matches appropriate features from query and stored images, calculates a similarity score between the query and each stored image examined, and displays the most similar images on the screen as thumbnails. The latest version of the system incorporates more efficient indexing techniques, an improved user interface, the ability to search grey-level images, and a video storyboarding facility.

VIRAGE

Another well-known commercial system is the VIR Image Engine from Virage, Inc [Gupta et al, 1996]. This is available as a series of independent modules, which systems developers can build in to their own programs. This makes it easy to extend the system by building in new types of query interface, or additional customized modules to process specialized collections of images such as trademarks. Alternatively, the system is available as an add-on to existing database management systems such as Oracle or Informix. An on-line demonstration of the VIR Image Engine. A high-profile application of Virage technology is AltaVista's AV Photo Finder , allowing Web surfers to search for images by content similarity. Virage technology has also been extended to the management of video data.

EXCALIBUR

A similar philosophy has been adopted by Excalibur Technologies, a company with a long history of successful database applications, for their Visual

RetrievalWare product [Feder, 1996]. This product offers a variety of image indexing and matching techniques based on the company's own proprietary pattern recognition technology. It is marketed principally as an applications development tool rather then as a standalone retrieval package. Its best-known application is probably the Yahoo! Image Surfer, allowing content-based retrieval of images from the World-wide Web**.**

VISUALSEEK

The VisualSEEk system [Smith and Chang, 1997a] is the first of a whole family of experimental systems developed at Columbia University, New York. It offers searching by image region colour, shape and spatial location, as well as by keyword. Users can build up image queries by specifying areas of defined shape and colour at absolute or relative locations within the image. The WebSEEk system [Smith and Chang, 1997b] aims to facilitate image searching on the Web. Web images are identified and indexed by an autonomous agent, which assigns them to an appropriate subject category according to associated text. Colour histograms are also computed from each image. At search time, users are invited to select categories of interest; the system then displays a selection of images within this category, which users can then search by colour similarity.

Similar Looks: We apply object detection to localize products such as bags and shoes. In this prototype, users click on automatically tagged objects to view similar-looking products.

SOME IMPORTANT TERMS :

1) **INDEXING :** Indexing a dataset is the process of quantifying our dataset by

utilizing an *image descriptor* to extract *features* from each image.

2) **IMAGE DESCRIPTOR :** An *image descriptor* defines the algorithm that we are utilizing to describe our image.

   For example:

   - The mean and standard deviation of each Red, Green, and Blue channel, respectively,
   - The statistical moments of the image to characterize shape.
   - The gradient magnitude and orientation to describe both shape and texture.

3) **FEATURES :** *Features*, on the other hand, are the output of an *image descriptor*. In the most basic terms, *features* (or *feature vectors*) are just a list of numbers used to abstractly represent and quantify images.

The pipeline of an image descriptor. An input image is presented to the descriptor, the image descriptor is applied, and a feature vector (i.e a list of numbers) is returned, used to quantify the contents of the image.

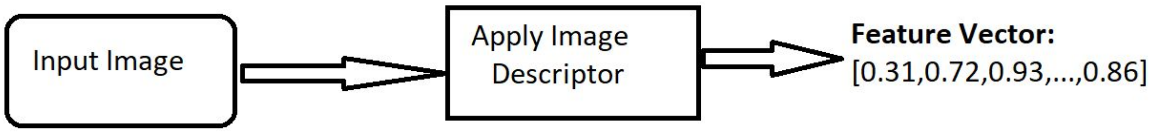


Fig 2- Image Descriptor

Here we are presented with an input image, we apply our image descriptor, and then our output is a list of features used to quantify the image.

Feature vectors can then be compared for similarity by using a ***distance metric*** or ***similarity function***. Distance metrics and similarity functions take two feature vectors as inputs and then output a number that represents how "similar" the two feature vectors are.

The figure below visualizes the process of comparing two images:

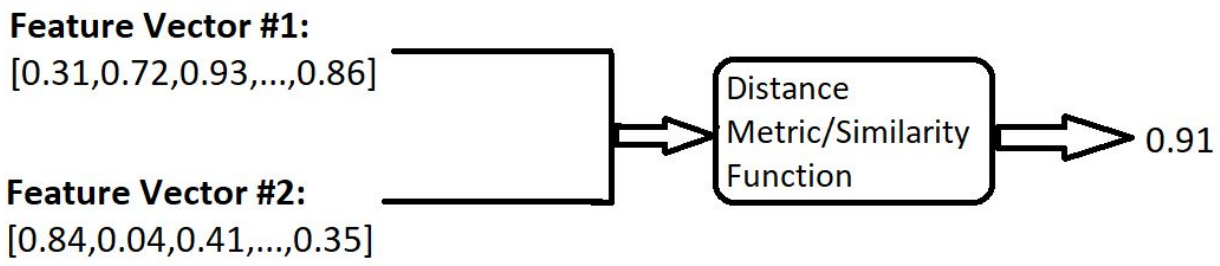
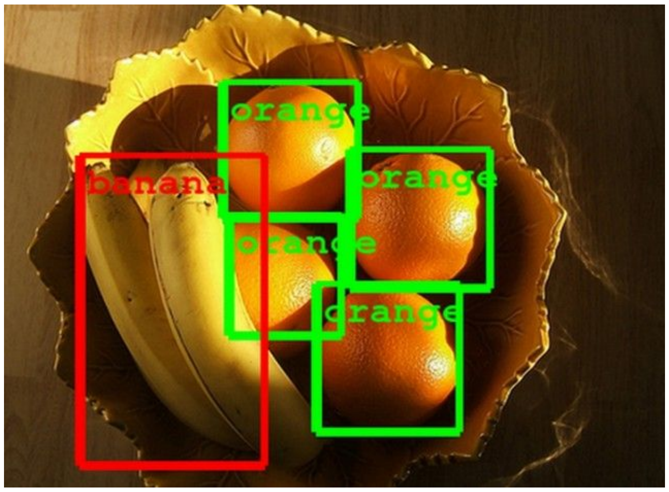


Fig. 3- Comparison of two images based on vector distance

To compare two images, we input the respective feature vectors into a distance metric/similarity function. The output is a value used to represent and quantify how “similar” the two images are to each other..

2.4 OBJECT DETECTION

It is the technology in the field of computer vision for finding and identifying objects in an image or video sequence. Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different viewpoints, in many different sizes and scales or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view. This task is still a challenge for computer vision systems. Many approaches to the task have been implemented over multiple decades.



APPLICATIONS :

Object recognition technology has matured to a point at which exciting applications are becoming possible. Indeed, industry has created a variety of computer vision products and services from the traditional area of machine inspection to more recent applications such as video surveillance, or face recognition.

Face detection

Popular applications include face detection and people counting. Have you ever noticed how facebook detects your face when you upload a photo? This is a simple application of object detection that we see in our daily life.

People Counting
Object detection can be also used for people counting, it is used for analysing store performance or crowd statistics during festivals. These tend to be more difficult as people move out of the frame quickly (also because people are non rigid objects).

Vehicle detection
Similarly when the object is a vehicle such as a bicycle or car, object detection with tracking can prove effective in estimating the speed of the object. The type of ship entering a port can be determined by object detection(depending on shape, size etc). This system for detecting ships are currently in development in some European countries.

Manufacturing Industry
Object detection is also used in industrial processes to identify products. Say you want your machine to only detect circular objects. Hough circle detection transform can be used for detection.

Online images
Apart from these object detection can be used for classifying images found online. Obscene images are usually filtered out using object detection.

Security
In the future we might be able to use object detection to identify anomalies in a scene such as bombs or explosives (by making use of a quadcopter).

3. <u>TECHNOLOGIES USED</u>

3.1 Python
Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant

whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software] and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:

Beautiful is better than ugly

- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

3.2 NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

3.3 CV2

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

Functions used in code :

1)urllib.request - The urllib.request module defines functions and classes which help in opening URLs (mostly HTTP) in a complex world — basic and digest authentication, redirections, cookies and more.

2)os.path - Unlike a unix shell, Python does not do any *automatic* path expansions. Functions such as expanduser() and expandvars() can be invoked explicitly when an application desires shell-like path expansion. (See also the glob module.)

3)Image.open - The Image module provides a class with the same name which is used to represent a PIL image. The module also provides a number of factory functions, including functions to load images from files, and to create new images.

Opens and identifies the given image file.

This is a lazy operation; this function identifies the file, but the file remains open and the actual image data is not read from the file until you try to process the data (or call the load() method). See new().

Parameters:	● **fp** – A filename (string), pathlib.Path object or a file object. The file object must implement **read()**, **seek()**, and **tell()** methods, and be opened in binary mode. ● **mode** – The mode. If given, this argument must be "r".
Returns:	An Image object.
Raises:	**IOError** – If the file cannot be found, or the image cannot be opened and identified.

4) plt.figure - matplotlib.pyplot.figure(*num=None*, *figsize=None*, *dpi=None*, *facecolor=None*, *edgecolor=None*, *frameon=True*, *FigureClass=<class 'matplotlib.figure.Figure'>*, *clear=False*, ***kwargs*)

num : integer or string, optional, default: none

If not provided, a new figure will be created, and the figure number will be incremented. The figure objects holds this number in a number attribute. If num is provided, and a figure with this id already exists, make it active, and returns a reference to it. If this figure does not exists, create it and returns it. If num is a string, the window title will be set to this figure's num.

figsize : tuple of integers, optional, default: None

width, height in inches. If not provided, defaults to rc figure.figsize**.**

4)plt.imshow() - matplotlib.pyplot.imshow(*X*, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, shape=None, filternorm=1, filterrad=4.0, imlim=None, resample=None, url=None, hold=None, data=None, **kwargs).

Display an image on the axes.

X : array_like, shape (n, m) or (n, m, 3) or (n, m, 4)

Display the image in X to current axes. X may be an array or a PIL image. If X is an array, it can have the following shapes and types:

- MxN – values to be mapped (float or int)
- MxNx3 – RGB (float or uint8)
- MxNx4 – RGBA (float or uint8)

MxN arrays are mapped to colors based on the norm (mapping scalar to scalar) and the cmap (mapping the normed scalar to a color).

Elements of RGB and RGBA arrays represent pixels of an MxN image. All values should be in the range [0 .. 1] for floats or [0 .. 255] for integers. Out-of-range values will be clipped to these bounds.

5) cv2.waitKey - It displays the image for specified milliseconds. Otherwise, it won't display the image. For example,waitKey(0) will display the window infinitely until any keypress (it is suitable for image display). waitKey(25) will display a frame for 25 ms, after which display will be automatically closed. (If you put it in a loop to read videos, it will display the video frame-by-frame).

You can use ord() function in Python for that.
For example, if you want to trigger 'a' keypress, do as follows :

```
if cv2.waitKey(33) == ord('a'): print "pressed a".
```

cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If **0** is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key a is pressed etc which we will discuss below.

6) cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.4 Tensorflow

TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as GPU cards. The system is flexible and can be used to express a wide variety of algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery. This paper describes the TensorFlow interface and an implementation of that interface that we have built at

Google. The TensorFlow API and a reference implementation were released as an open-source package under the Apache 2.0 license.

3.5 PYTHON VS OTHER LANGUAGES FOR OBJECT DETECTION

Object detection is a domain-specific variation of the machine learning prediction problem. Intel's OpenCV library that is implemented in C/C++ has its interfaces available in a number of programming environment such as C#, Matlab, Octave, R, Python etc. Some of the benefits of using Python codes over other language codes for object detection are:

- More compact and readable code.
- Python uses zero-based indexing.
- Dictionary (hashes) support is offered.
- Simple and elegant.
- Object-oriented programming.

4. IMPLEMENTATION OF CBIR USING PYTHON

In CBIR , we had created a retrieval system that used the content of image. The user had to provide a query image , color histogram of the query image would be calculated and the most closest images from the database would be retrieved .

4.1 The Four Steps of Any CBIR System

Any Content Based Image Retrieval System has four basic steps -

1. **Defining the Image Descriptor :** We need to decide what aspect of the image we want to describe namely - the colour of the image , the shape or the texture of the image.
2. **Indexing the Dataset :** Now that we have the image descriptor defined,we need to apply this image descriptor to each image in the dataset, extract

features from these images, and write the features to storage (ex. CSV file, RDBMS, Redis, etc.) so that they can be later compared for similarity.

3. **Defining the similarity metric :** After indexing the dataset , we now have a bunch of feature vectors. We need to compare these feature vectors. Popular choices include the Euclidean distance, Cosine distance, and chi-squared distance, but the actual choice is highly dependent on (1) the dataset and (2) the types of features that we have extracted.
4. **Searching :** The final step is to perform an actual search. A user will submit a query image and we have to (1) extract features from this query image and then (2) apply the similarity function to compare the query features to the features already indexed.Then, the most relevant results are returned according to the similarity function.

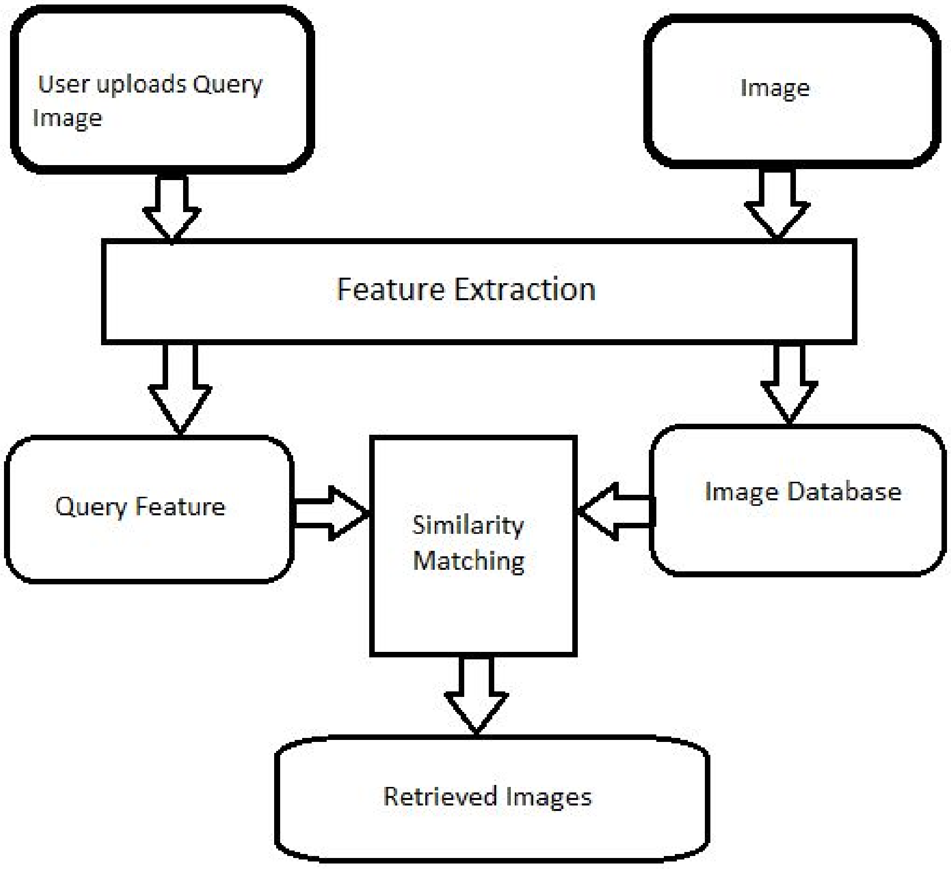


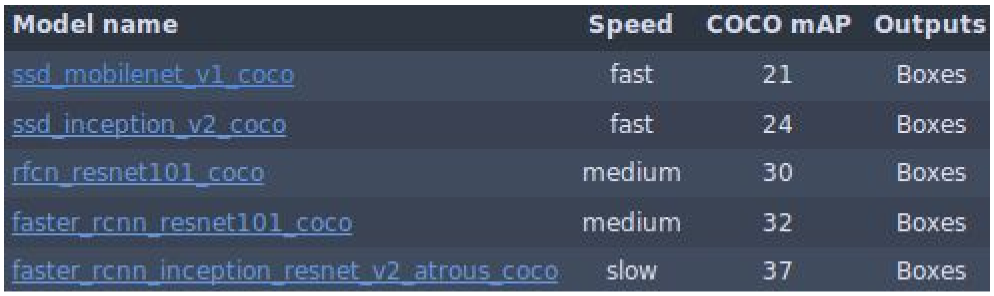
Fig. 4-Image Extraction Architecture

5. IMPLEMENTATION OF OBJECT DETECTION

In Object Detection , a particular object will be detected from whole image. For example, from an image,consisting of several animals, we want to find particularly image of a cat, then a set of images of cat will be stored and image detector will be trained. It is a part of computer vision.

5.1 Selecting a Model

The default model in the notebook is the simplest (and fastest) pre-trained model offered by TensorFlow. Looking at the table below, you can see there are many other models available. mAP stands for mean average precision, which indicates how well the model performed on the COCO dataset. Generally models that take longer to compute perform better.



Model name	Speed	COCO mAP	Outputs
ssd_mobilenet_v1_coco	fast	21	Boxes
ssd_inception_v2_coco	fast	24	Boxes
rfcn_resnet101_coco	medium	30	Boxes
faster_rcnn_resnet101_coco	medium	32	Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	slow	37	Boxes

FIG 5: Models in TensorFlow API

To get a rough approximation for performance just try each model out on a few sample images. If the item you are trying to detect is not one of the 90 COCO classes, find a similar item (if you are trying to classify a squirrel, use images of small cats) and test each model's performance on that.

5.2 COCO DATASET

COCO is a large image dataset designed for object detection, segmentation, person keypoints detection, stuff segmentation, and caption generation.COCO dataset contains photos of 91 objects types that would be easily recognizable by a 4 year old. With a total of 2.5 million labeled instances in 328k images, the creation of our dataset drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting and instance segmentation.

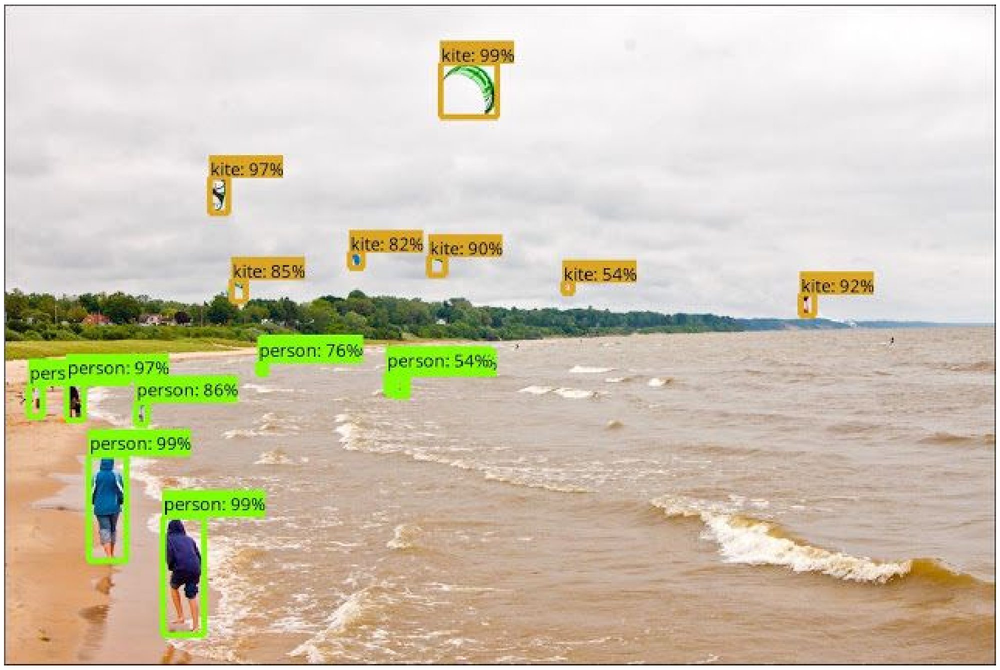


Fig 6 : Object Detection using TensorFlow

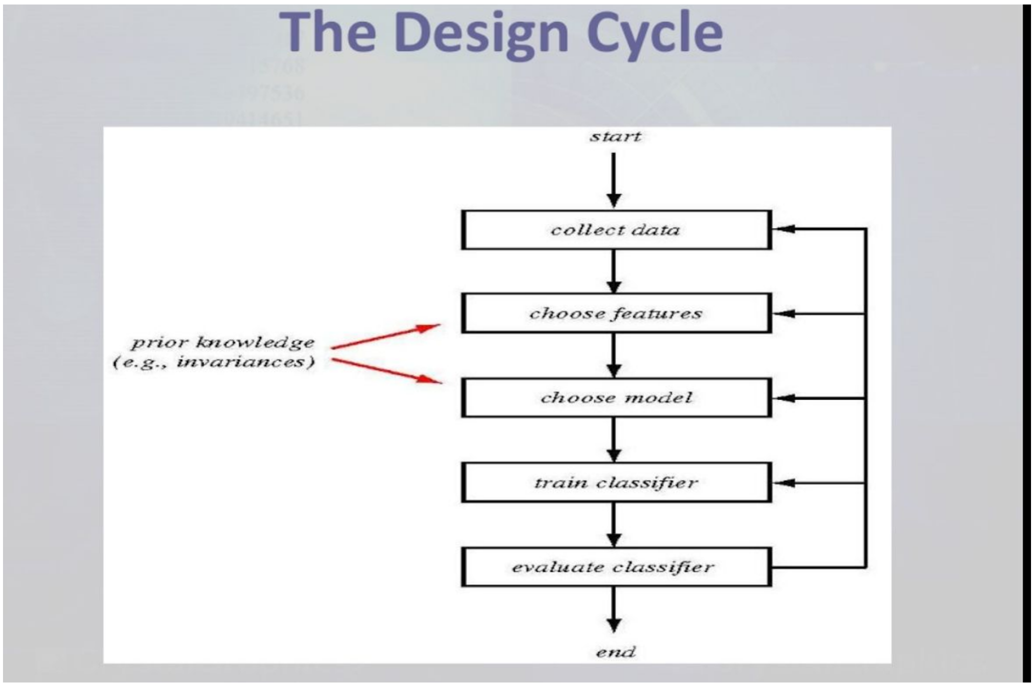


Fig 7 : **Design Cycle**

Code For CBIR

search.py

```
from pyimagesearch.colordescriptor import ColorDescriptor
from pyimagesearch.searcher import Searcher
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--index", required = True,
	help = "Path to where the computed index will be stored")
ap.add_argument("-q", "--query", required = True,
	help = "Path to the query image")
ap.add_argument("-r", "--result-path", required = True,
	help = "Path to the result path")
args = vars(ap.parse_args())

# initialize the image descriptor
cd = ColorDescriptor((8, 12, 3))
query = cv2.imread(args["query"])
features = cd.describe(query)


searcher = Searcher(args["index"])
results = searcher.search(features)

cv2.imshow("Query", query)

for (score, resultID) in results:
	# load the result image and display it
	result = cv2.imread(args["result_path"] + "/" + resultID)
	cv2.imshow("Result", result)
	cv2.waitKey(0)
```

OUTPUT

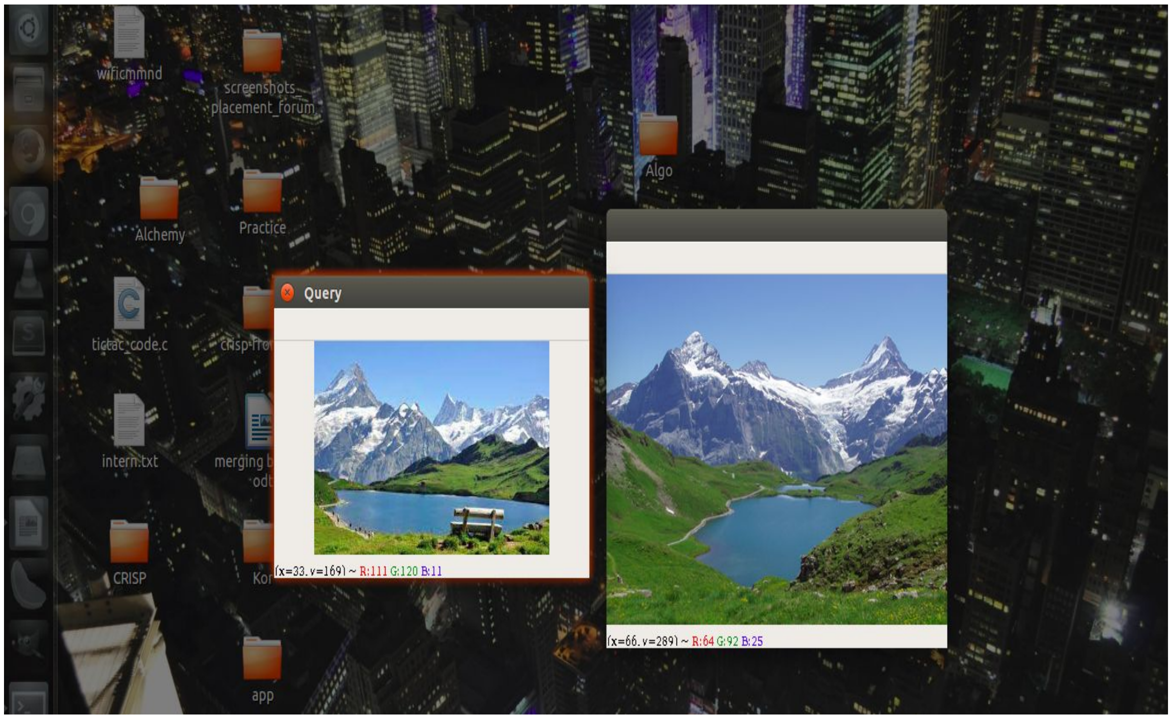


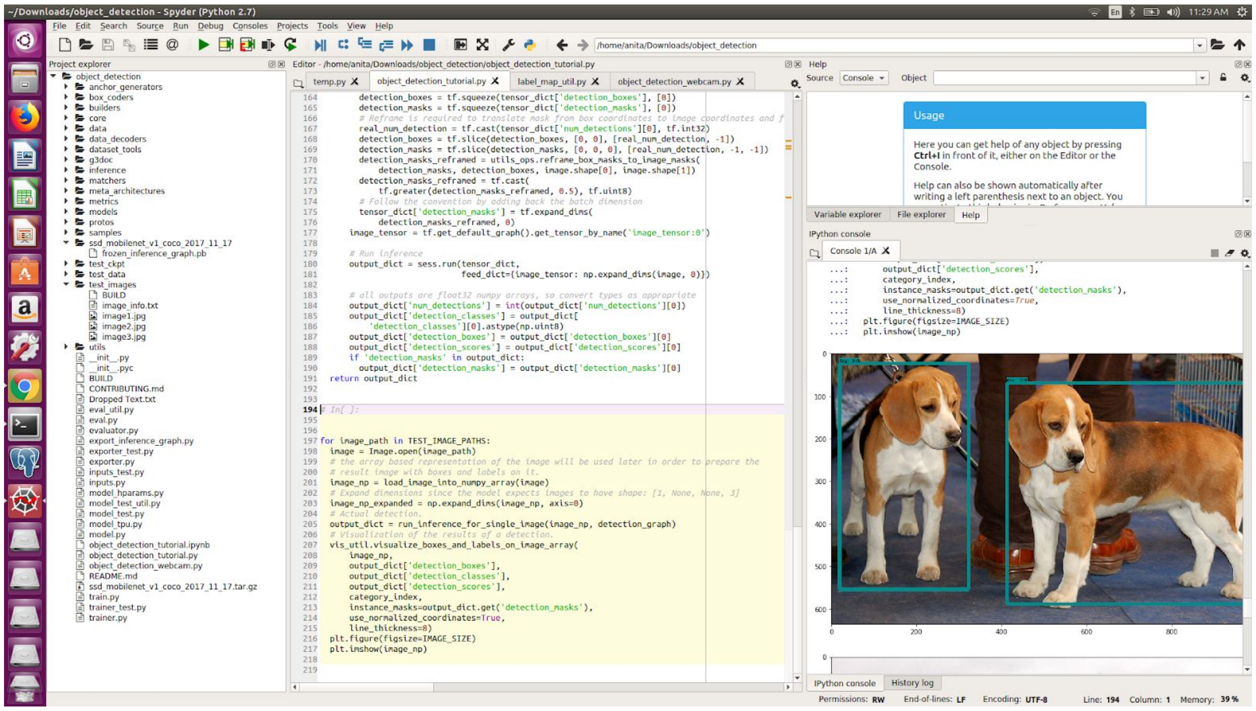
Fig 8- Content Based Image Retrieval

Code : Object Detection in Images

```
import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile
import tensorflow as tf

import zipfile

from collections import defaultdict

from io import StringIO

from matplotlib import pyplot as plt

from PIL import Image


# This is needed since the notebook is stored in the object_detection folder.

sys.path.append("..")

from utils import ops as utils_ops


if tf.__version__ < '1.4.0':

  raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')


# This is needed to display the images.

get_ipython().magic(u'matplotlib inline')


from utils import label_map_util

from utils import visualization_utils as vis_util
```

```
# What model to download.

MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'

MODEL_FILE = MODEL_NAME + '.tar.gz'

DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'


# Path to frozen detection graph. This is the actual model that is used for the object
detection.

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'


# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')


NUM_CLASSES = 90


opener = urllib.request.URLopener()

opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar_file = tarfile.open(MODEL_FILE)

for file in tar_file.getmembers():

  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())


detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')


label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories        =        label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)



def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
   (im_height, im_width, 3)).astype(np.uint8)


PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,
'image{}.jpg'.format(i)) for i in range(1, 4) ]


# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
            tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
            # Reframe is required to translate mask from box coordinates to image
coordinates and fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1,
-1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
          detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
          tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
          detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
    return output_dict

for image_path in TEST_IMAGE_PATHS:
    image = Image.open(image_path)
    # the array based representation of the image will be used later in order to prepare
the
```

```
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
   # Expand dimensions since the model expects images to have shape: [1, None,
None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```



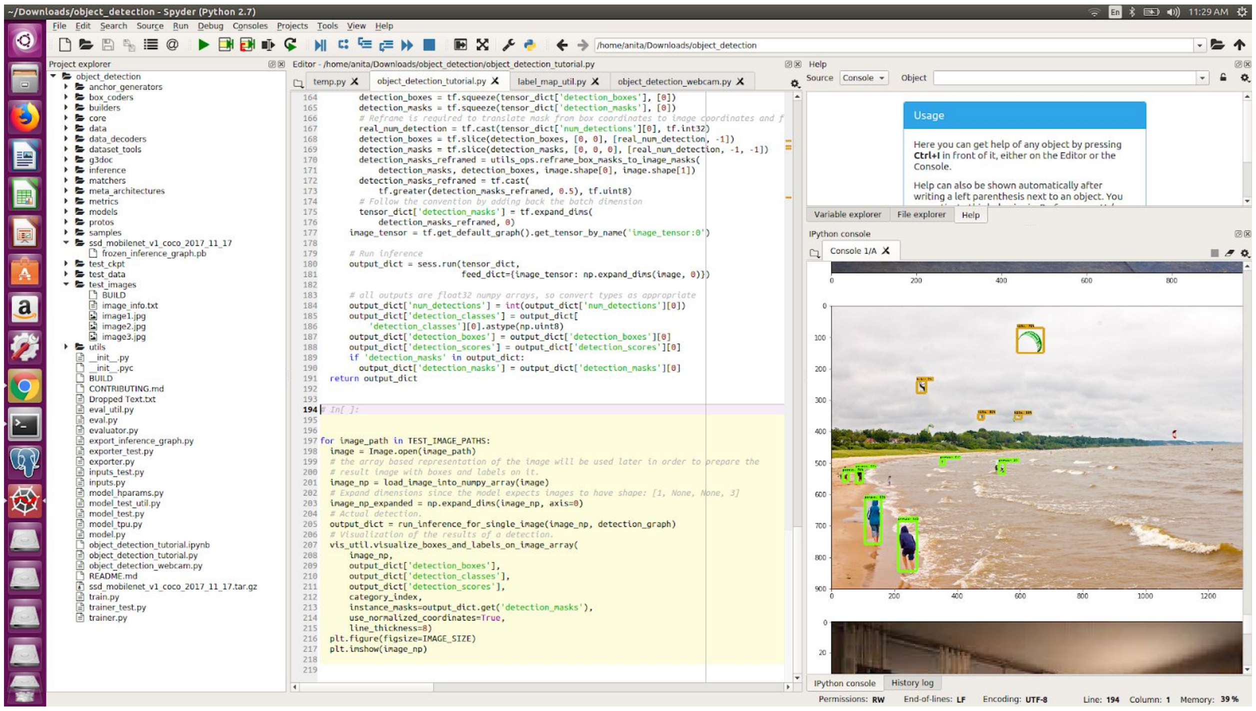
```
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinates and f
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict


# In[ ]:


for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You

```
   ...:         output_dict['detection_scores'],
   ...:         category_index,
   ...:         instance_masks=output_dict.get('detection_masks'),
   ...:         use_normalized_coordinates=True,
   ...:         line_thickness=8)
   ...:     plt.figure(figsize=IMAGE_SIZE)
   ...:     plt.imshow(image_np)
```

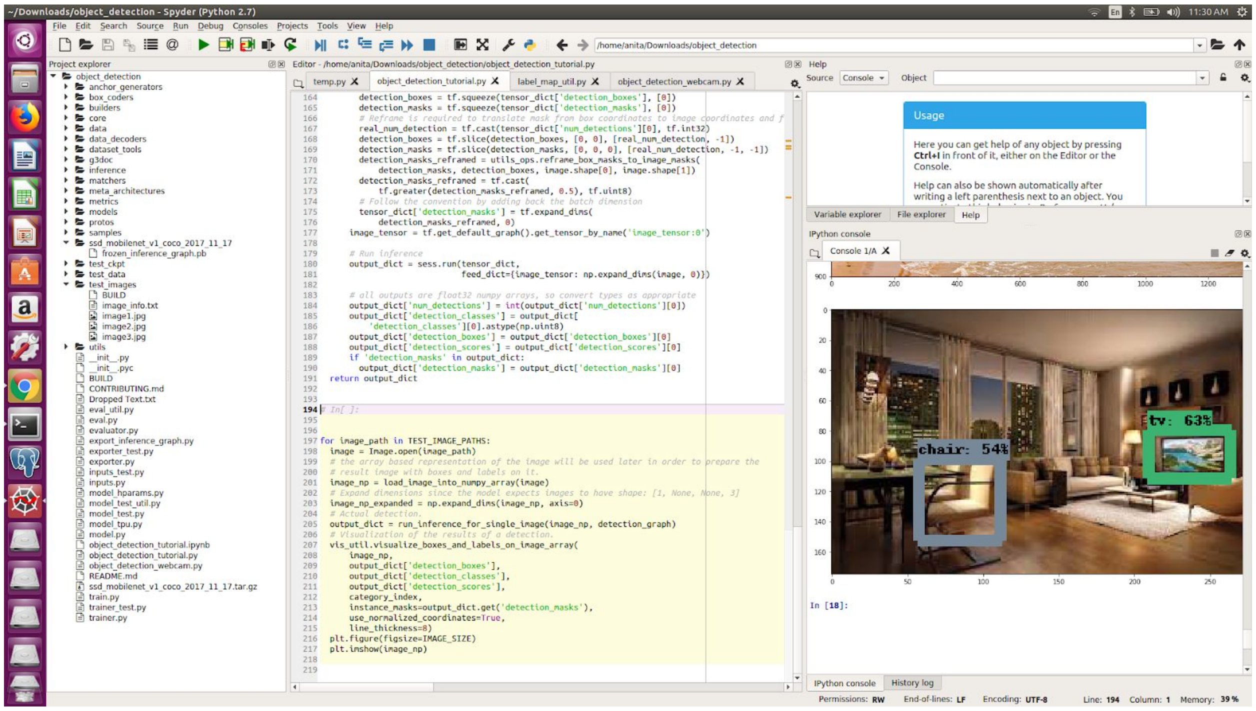


```
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinates and f
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict


# In[ ]:


for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You


Editor - /home/anita/Downloads/object_detection/object_detection_tutorial.py

```
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinates and f
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict


# In[ ]:


for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You

~/Downloads/object_detection - Spyder (Python 2.7)
temp.py
object_detection_tutorial.py
label_map_util.py
object_detection_webcam.py
chair: 54%
tv: 63%
In [18]:

Code for object detection using webcam :

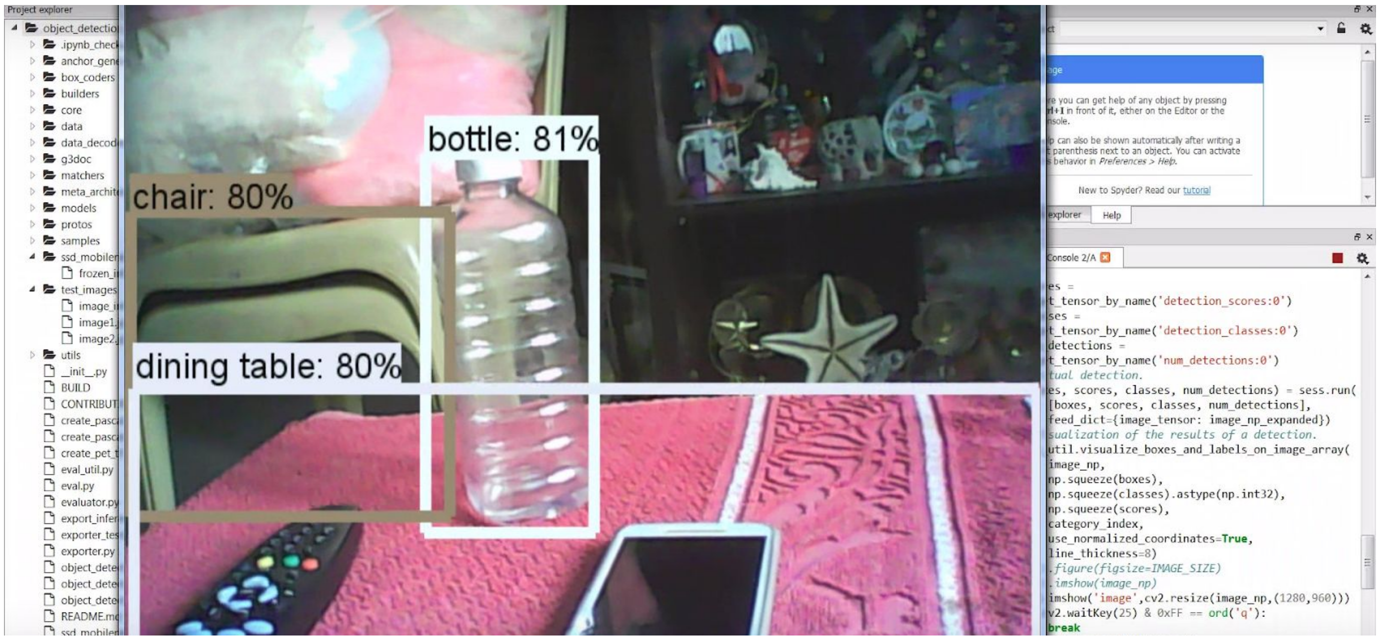
```
import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile

import tensorflow as tf

import zipfile

from collections import defaultdict
from io import StringIO

from matplotlib import pyplot as plt

from PIL import Image


sys.path.append("..")

from utils import ops as utils_ops


if tf.__version__ < '1.4.0':

  raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')


# This is needed to display the images.

get_ipython().magic(u'matplotlib inline')


from utils import label_map_util


from utils import visualization_utils as vis_util

# What model to download.

MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'

MODEL_FILE = MODEL_NAME + '.tar.gz'

DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
```

```
# Path to frozen detection graph. This is the actual model that is used for the object detection.

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'


# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')


NUM_CLASSES = 90


opener = urllib.request.URLopener()

opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar_file = tarfile.open(MODEL_FILE)

for file in tar_file.getmembers():

  file_name = os.path.basename(file.name)

  if 'frozen_inference_graph.pb' in file_name:

    tar_file.extract(file, os.getcwd())



# ## Load a (frozen) Tensorflow model into memory.
# In[ ]:



detection_graph = tf.Graph()

with detection_graph.as_default():

  od_graph_def = tf.GraphDef()

  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:

    serialized_graph = fid.read()

    od_graph_def.ParseFromString(serialized_graph)

    tf.import_graph_def(od_graph_def, name='')


label_map = label_map_util.load_labelmap(PATH_TO_LABELS)

categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)

category_index = label_map_util.create_category_index(categories)



# ## Helper code
# In[ ]:



def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)


# # Detection


# In[ ]:



# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the images to the
TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
```

```
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,
'image{}.jpg'.format(i)) for i in range(1, 3) ]


# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)


def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
          tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
            # Reframe is required to translate mask from box coordinates to image
coordinates and fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1,
-1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
           detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
           tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
           detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')
```

```
      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict



# In[ ]:


import cv2
cap = cv2.VideoCapture(0)
ret = True
```

```
#for image_path in TEST_IMAGE_PATHS:
while(ret):
#  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
  # result image with boxes and labels on it.
#  image_np = load_image_into_numpy_array(image)
  ret,image_np = cap.read()
  # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
        instance_masks=output_dict.get('detection_masks'),
        use_normalized_coordinates=True,
        line_thickness=8)
#  plt.figure(figsize=IMAGE_SIZE)
#  plt.imshow(image_np)
  cv2.imshow('image',cv2.resize(image_np,(1280,960)))
  if cv2.waitKey(25)&0xFF == ord('q'):
      break
      cv2.destroyAllWindows()
      cap.release()
```



re you can get help of any object by pressing
rl+I in front of it, either on the Editor or the
nsole.

lp can also be shown automatically after writing a
t parenthesis next to an object. You can activate
s behavior in *Preferences > Help*.

```
es =
t_tensor_by_name('detection_scores:0')
ses =
t_tensor_by_name('detection_classes:0')
detections =
t_tensor_by_name('num_detections:0')
tual detection.
es, scores, classes, num_detections) = sess.run(
[boxes, scores, classes, num_detections],
feed_dict={image_tensor: image_np_expanded})
sualization of the results of a detection.
util.visualize_boxes_and_labels_on_image_array(
image_np,
np.squeeze(boxes),
np.squeeze(classes).astype(np.int32),
np.squeeze(scores),
category_index,
use_normalized_coordinates=True,
line_thickness=8)
.figure(figsize=IMAGE_SIZE)
.imshow(image_np)
imshow('image',cv2.resize(image_np,(1280,960)))
v2.waitKey(25) & 0xFF == ord('q'):
break
```

5. Conclusion

The research in this field has come a long way during the last decade, but it has still a long way to go to provide the users with tools to retrieve images from the multimedia or image databases in a very efficient way through input in the form of text, image or drawing.

With the increase in the number of digital devices being used and the growth of Internet, thousands of images are being added daily in the image database . The need for efficient retrieval of an image from a large collection is shared by many professionals including design engineers, journalists etc.

It has been acknowledged that there always remains a room for improvement in currently used systems.The images need to be stored and retrieved in an effective and efficient manner. Also, searching time is most important for any search method while searching in a large dataset. So, by using '*Content Based Image Retrieval'* image retrieval will be an efficient process.

There is overgrown possibilities that computer vision can be used to solve the real time problems. It has given machines an artificial intelligence and human like vision. The essentials of object detection along with different methods for accomplishing it and its extension has been
examined. Using python for the implementation ,over MATLAB is preferred because of its easiness and correctness.

6. References :

Paper presented in IndiaCom 2018 - **"Content Based Image Retrieval Using Hadoop "**

[1]R. Priyatharshini and S. Chitrakala, "Association based image retrieval: A survey, Springer-Verlag Berlin Heidelberge, pp. 17-26, 2013.

[2]Y. Rui and T. S. Huang, "Image retrieval: Current techniques, promising directions and open issues," Journal of Visual Communication and Image Representation,vol. 10, pp. 39{62, 1999.

[3] A. Ponomarev et al., "Content-Based Image Retrieval Using Color, Texture and Shape Features", Key Engineering Materials, Vol. 685, pp. 872-876, 2016

[4,1]Yushi Jing, David Liu , Dmitry Kislyuk , Andrew Zhai , Jiajing Xu , Jeff Donahue, Sarah Tavel "Visual Search at Pinterest " 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015

[5] U.S.N. Raju, Shibin George, V. Sairam Praneeth, Ranjeet Deo,Priyanka Jain Department of Computer Science and Engineering, National Institute of Technology, Warangal, India -*"Content Based Image Retrieval on Hadoop Framework"*

[6] Nima Razavi, Juergen Gall and Luc Van Gool, "Scalable Multi-class Object Detection", IEEE Conference on Computer Vision and Pattern Recognition, pp. 1505- 1512, 2011.

[7]Swati V. Sakhare & Vrushali G. Nasre *," Design of Feature Extraction in Content Based Image Retrieval(CBIR) using Color and Texture.*

[8] Shankar M. Patil "Content Based Image Retrieval Using Color, Texture and Shape," International Journal of Computer Science & Engineering Technology (IJCSET), Vol. 3, Sept. 2012.

[9] Ren, F.X., Huang, J.S., Jiang, R.Y., Klette, R.: General Traffic Sign Recognition by Feature Matching. In: IVCNZ, pp. 409–414 (2009)

7. Appendix

Object Detection And Image Recognition

Arun Kumar Dubey[1] ,Anita Kumari[2], Palak Garg[3], Anuradha R Pai[4]
[1,2,3,4]Bharati Vidyapeeth's College of Engineering, New Delhi
arudubey@gmail.com

Abstract

With the tremendous increase in the amount of images that are being produced daily , there was a need for the development of an robust and efficient object detection system.Object recognition is the process by which objects are detected within images and videos.
Object detection can be used for various purposes including retrieval and surveillance. In this study, various basic concepts used in object detection are described while making use of OpenCV library of python 2.7, improving the efficiency and accuracy of object detection are presented.
Keywords: Object Detection,Python, OpenCV,Numpy,Haar Cascade

1. Introduction

The modern world is encircled with heavy masses of digital visual information, may it be images,videos and so on. It is a need of hour to analyse and organise these plundering ocean of visual information and for these techniques are required. Particularly,it will be more useful to analyse semantic information of the images or videos.One imperative part of image content is the objects in the image. So there is a need for object detection and image recognition techniques.
Object detection is a vital, yet difficult vision task. It is a basic part in numerous

applications, for example, image search, image auto-annotation and scene understanding;
be that as it may it is as yet an open issue because of the intricacy of object classes and images. It is being widely used in industries to ease user, save time and to achieve parallelism. Object detection is a part of computer vision which aims at having a human like vision,which can locate and differentiate various objects such as, numbers,location,size, position etc. The common object detection method is the color-based approach, detecting objects based on their color values.
Object detection is one of the most challenging applications of the image processing. It is a branch of computer vision and artificial intelligence. The aim of this project is to identify and locate the objects in the images or videos and also naming the specific objects detected.
OpenCv(python) is used for the implementation of the project. Numpy library is also required for the same. Also, an OpenCv algorithm i.e Haar Cascade is used and Haar like features are emphasized.

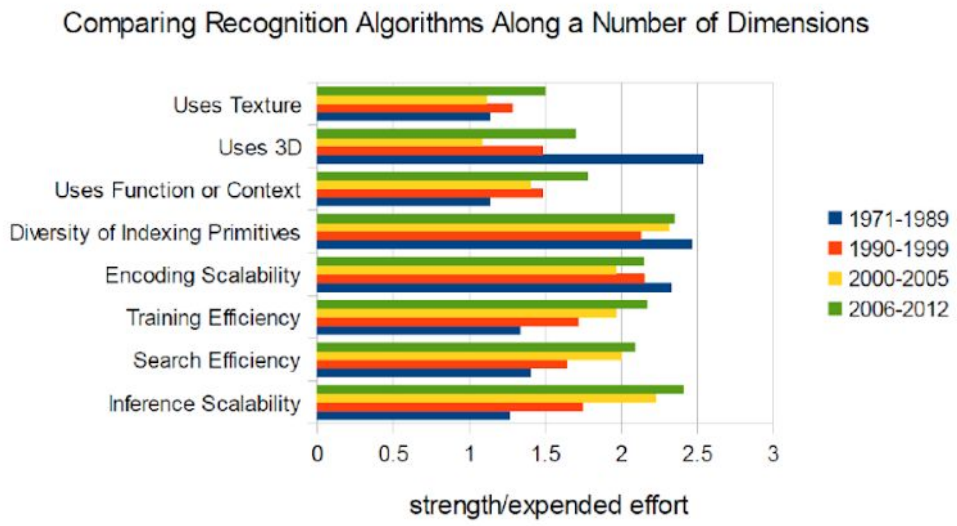
Computer Vision

Humans use their eyes and brain to see and spot the objects around.Computer vision is the science giving similar functionality and capability to a machine or a computer. It aims at enabling computers to see,identify and process the images in the same way as human vision does, and then provide appropriate output. It resembles giving human intelligence and instincts to a computer.

2. Background

Modern computer vision approach has its starting point in early 1960s. First and foremost application was pattern recognition systems used in offices. While performing a task,some practical difficulties such as scene complexities increased as some illumination variability increased. Also time, cost and sensor noise constraints became for common.
1964 involved the automation of the wire-bonding process of transistors, with the ultimate goal of replacing human workers. However ,it attained 95% accuracy in test labs but regarded too low to replace human workers.By 1973, fully automated

assembly machines had been made , resulting in the world's first image-based machine for the automatic assembly of semiconductor devices. It is much more evolved in these 50 years and achieved almost human accuracy.
As the experiments went on and on making the systems more efficient and reducing the need of human workers, computer vision has now become an effective and efficient technology. working very much like human vision. It is now being used in various fields like face recognition,autonomous cars,robots and ofcourse object detection and recognition.



3. Description of tools

In this section, tools and methods which are used for the object detection and image recognition process are described.OpenCV and Numpy are the two python libraries which are required and used for the implementation of the same.

3.1 OpenCv

OpenCv(Open Source Computer Vision) is a computer vision and machine learning software library.It was initially built to provide a common infrastructure for applications related to computer vision and to increase the use of machine perception in the commercial products. This library contains around 3000

algorithms inside and each of them is highly optimized. It mounts real-time vision applications.
These algorithms are easily implemented in Java, MATLAB, Python, C, C++ etc. and are well supported by operating system like Window, Mac OS, Linux and Android. It is free for use under open source BSD license. Initially,this library was written in C and this C interface made OpenCV movable to some specific platforms such as digital signal processors. Wrappers for languages such as C#, Python, Ruby and Java (using JavaCV) have been developed to encourage adoption by a wider audience.
It works well with python 2.7 but first Numpy package has to be installed.

3.2Numpy

Numpy is an essential python package which is used for scientific computations. It supports multidimensional arrays and matrices. It contains a powerful N-dimensional array object,sophisticated (broadcasting) functions,tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities. Not only these scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Therefore , it can be used to store the corresponding RGB or HSV values of the image objects in this multi-dimensional container. It is licensed under BSD license and hence, can be reused with few restrictions.

4. OpenCv(Python) Vs MATLAB

Using OpenCv in computer vision has many pros as compared to that of MATLAB.

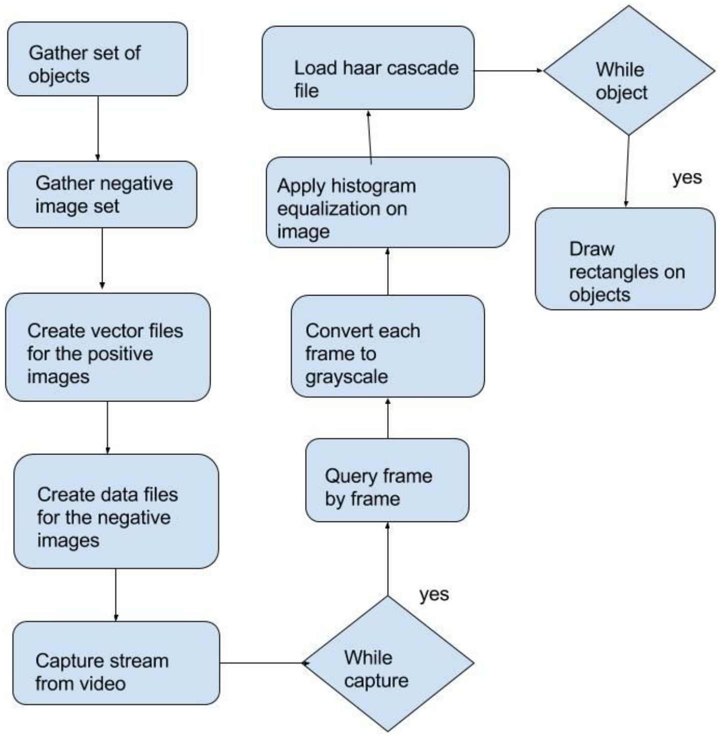
I. **Ease of Use:** Using and learning python programming is very easy. Python is a simple language and can be learnt in a short duration of time even ,if not known before. Whereas, there is a different way of writing codes in MATLAB.
II. **Python is new scientific computing language:** Pastly, MATLAB was known as the scientific computing language but now, with libraries like OpenCV,matplotlib,Numpy it has become more efficient.

III. **Less implementation cost:** As compared to MATLAB, implementation cost of the project in python is much lesser.

IV. **Visualization and Debugging:** Visualisation with matplotlib is as effective as MATLAB . Also, debugging is much easier in python as compared to other languages.

V. **Runtime:** MATLAB has slower runtime that of Python.

5. Implementation of the Model



Gather set of objects
Gather negative image set
Create vector files for the positive images
Create data files for the negative images
Capture stream from video
While capture
yes
Query frame by frame
Convert each frame to grayscale
Apply histogram equalization on image
Load haar cascade file
While object
yes
Draw rectangles on objects

An object detection and image recognition model follows broadly three basic steps that are:

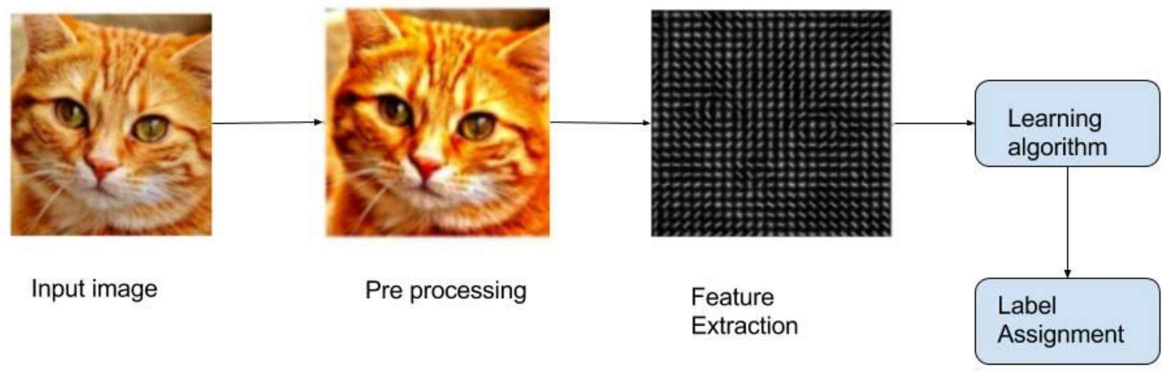
A. Image Classification
B. Detection

A. Image Classification

An image classifier follows an algorithm which takes an image as input and outputs what the image contains. The output is the image label which tells about the class to which the image belongs to(for example, cat,elephant,chair etc). But for recognising the content of the image, image classifier has to be trained well. It is ,therefore, trained by giving different set of positive and negative images to differentiate among different features and to give the desired output.

Anatomy of an Image Classifier

The following diagram shows how an image classifier works:



Step 1: Preprocessing

Firstly an input picture is pre-processed to normalize the contrast and brightness effects.Also, preprocessing step involves subtracting the mean of image intensities and divide by the standard deviation. Methods like gamma correction and colour space transformation can also be used for the same and may result in better results ,sometimes.

After that, the input image is cropped and resized to a fixed size image as it is integral part for the feature extraction.

Step 2: Feature Extraction

It is very important to remove the extra information contained in the image that is not necessary for the classification. Therefore,firstly simplification of the image by extracting only important information and leaving the rest is done. It can be done by running an edge detector on the image.Some well-known features used in computer vision are Haar-like features introduced by Viola and Jones, Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Feature (SURF) etc.

We will be using Haar-like features for the implementation of our model.

Haar-like features

It is an OpenCv algorithm based on machine learning approach where a cascade function is trained by a lot of positive and negative images. After input of positive and negative images, features are being extracted based on Haar features shown in the figure below.

Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. Now all possible sizes and locations of each kernel is used to calculate overflowing features. To make is less cumbersome integral images were introduced which simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. It surely makes the process super fast but also, extract many irrelevant features. So for extracting the best and important features from those 160000+ features, *Adaboost* works well.
Each and every feature is applied in all the training images and best threshold which will classify the object is found. There will surely be errors and misclassifications but we choose the one with minimum error rate, which means they are the best features which classifies the object best. The process is continued to achieve new error rates and weights to meet the accuracy.Final classifier is a weighted sum of these weak classifiers.
Now, the final setup has around 6000 features. By *Cascade Classifier* , these 6000 features are applied on different stages. If a window fails the first stage, it is discarded and if not it goes to the next stage. It saves a lot of time as comparing 6000 features at a time.

B. Detection

For the detection purpose,

I. A sliding window is performed over the image,i.e, it moves through the whole image and at different scales.
II. For each windows of the sliding window, features are extracted, computed and classified.

III. A rectangle is drawn and the name is flashed on the object being detected.

6.STEPS INVOLVED IN OBJECT DETECTION IN PYTHON 2.7

6.1 Installation of OpenCV-python

For installation of OpenCv, it is required to install the packages-python 2.7.x, Numpy,matplotlib to their default locations. Python will be installed to c/Python27/. On the terminal, import Numpy and make sure it is working well. After that OpenCv is downloaded Go to OpenCV/build/python/2.7 folder. Copy cv2.pyd to C:/Python27/lib/site-packages.

6.2 Read an Image

To read an image,CV2.imread() function is used. The image should be in the same directory,otherwise, full path of the image should be entered. Arguments specifying how an image should be read are:

1. CV2.IMREAD_COLOR: It is the default flag used to load the colors of the image. It tells about the transparency of the image.
2. CV2.IMREAD_GRAYSCALE: This function is used to present the image in grayscale mode for extraction of the features.
3. CV2.IMREAD_UNCHANGED: Loads image as such including alpha channel.

6.3 Feature Detection and Description

In this step, understanding of the features and extraction of the required features from the images is done. For this,OpenCV provides two techniques, Brute-Force matcher, and FLANN based matcher.

7. Mathematical Description

HAAR LIKE FEATURES

Haar detection Cascade uses the idea of eliminating the negative images with very little processing. Number of classifiers are computed using OpenCV to every sub-region in the image. On the off chance that the sub-region does not pass the majority of the classifiers than that picture is discarded and further calculation is performed. On the other hand if the sub-region passes the first stage then it is

passed onto the next stage for further calculations. For the success of object detection, an image sub-region must pass all the classifiers. Classifier can be trained using OpenCV.
Haar-like features are an over entire arrangement of two-dimensional (2D) Haar capacities, which can be utilized to encode nearby appearance of articles.

8.Applications

8.1 Face detection

Well known applications incorporate face recognition and individuals tallying. Have you at any point seen how facebook recognizes your face when you upload a photograph? This is a straightforward use of object detection that we find in our day by day life.

8.2 People Counting

People counting can be done with the help of Object detection. Analysing store performance or crowd statistics during festivals can also be done using object detection. These have a tendency to be more troublesome as individuals move out of the frame quickly.

8.3 Vehicle detection

Object detection is useful in detecting vehicles such as bicycle or truck. It is also effective in estimation of the object’s speed. Object detection can also help in determining the type of ship entering a port. Some of the European countries are using this method for detecting ships.

8.4 Manufacturing Industry

Object Detection is additionally utilized as a part of modern procedures to distinguish items. Let's assume you need your machine to just recognize round articles. Hough circle discovery change can be utilized for identification.

8.5 Online images

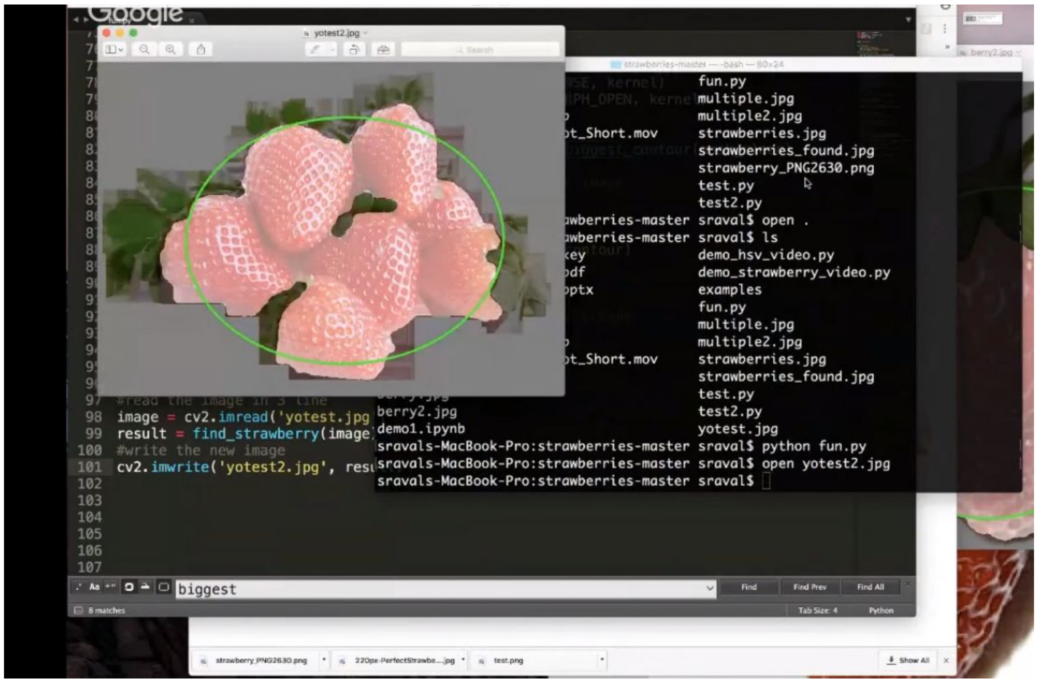
Aside from these object detection can be utilized for grouping pictures discovered on the web. Vulgar pictures are typically filtered out utilizing object detection.

8.6 Security

Later on we may have the capacity to utilize object detection to recognize abnormalities in a scene, for example, bombs or explosives (by making utilization of a quadcopter).

9. Result and Analysis

Our project detect different objects including strawberries,bike,car, duck and many more. The figure below shows the result generated after performing search for strawberries in a query image. The circled part is the final result.



```
image = cv2.imread('yotest.jpg
result = find_strawberry(image
#write the new image
cv2.imwrite('yotest2.jpg', resu
```

```
                                             fun.py
                                             multiple.jpg
                                             multiple2.jpg
                                             strawberries.jpg
                                             strawberries_found.jpg
                                             strawberry_PNG2630.png
                                             test.py
                                             test2.py
awberries-master sraval$ open .
awberries-master sraval$ ls
key                                          demo_hsv_video.py
odf                                          demo_strawberry_video.py
optx                                         examples
                                             fun.py
                                             multiple.jpg
                                             multiple2.jpg
ot_Short.mov                                 strawberries.jpg
                                             strawberries_found.jpg
                                             test.py
berry2.jpg                                   test2.py
demo1.ipynb                                  yotest.jpg
sravals-MacBook-Pro:strawberries-master sraval$ python fun.py
sravals-MacBook-Pro:strawberries-master sraval$ open yotest2.jpg
sravals-MacBook-Pro:strawberries-master sraval$
```

10. Conclusion

There is overgrown possibilities that computer vision can be used to solve the real time problems. It has given machines an artificial intelligence and human like vision. The essentials of object detection along with different methods for accomplishing it and its extension has been
examined. Using python for the implementation ,over MATLAB is preferred because of its easiness and correctness.
Two major steps in object detection are feature understanding and matching and it is necessary that they are performed with high accuracy. Haar cascade algorithm of openCv is being used for completion of the project.
OpenCv will surely acquire an important place in the IT companies and will be a prime requirement for the coders,in coming years.