|  |  |  |
| --- | --- | --- |
| Ch.  No. | Contents | Page No. |
| 1 | Introduction |  |
|  | 1.1. Project Purpose | 2 |
|  | 1.2. Project Scope | 3 |
|  | 1.3. Project Objectives | 3 |
|  | 1.4. Project Goals | 4 |
| 2 | System Requirement Specifications |  |
|  | 2.1. Scope | 5 |
|  | 2.2. Objectives | 6 |
|  | 2.3. Product Description | 6 |
|  | 2.4. Specific Requirements | 7 |
|  | 2.5. Functional and Non-Functional Requirements | 8 |
|  | 2.6. System Attributes | 8 |
| 3 | System Design |  |
|  | 3.1. Use Case Diagram | 10 |
|  | 3.2. Sequence Diagram | 14 |
|  | 3.3. Activity Diagram | 15 |
|  | 3.4. Data Flow Diagram | 16 |
|  | 3.5. E-R Diagram | 20 |
|  | 3.6. Data Modelling | 21 |
| 4 | Source Code | 30 |
| 5 | Output Screens | 66 |
| 6 | Testing | 76 |
| 7 | Conclusion and Discussion |  |
|  | 7.1. Future Scope of the Project | 85 |
|  | 7.2. Self-analysis of project viabilities | 85 |
|  | 7.3. Problem encountered and possible solutions | 85 |
|  | 7.4. Summary of the project work | 85 |

# CHAPTER 1

**PROJECT INTRODUCTION**

### Objective

The goal of “object detection” is to find the location of an object in a given picture accurately and mark the object with the appropriate category. To be precise, the problem that object detection seeks to solve involves determining where the object is, and what it is. However, solving this problem is not easy. Unlike the human eye, a computer processes images in two dimensions. Furthermore, the size of the object, its orientation in the space, its attitude, and its location in the image can all vary greatly.

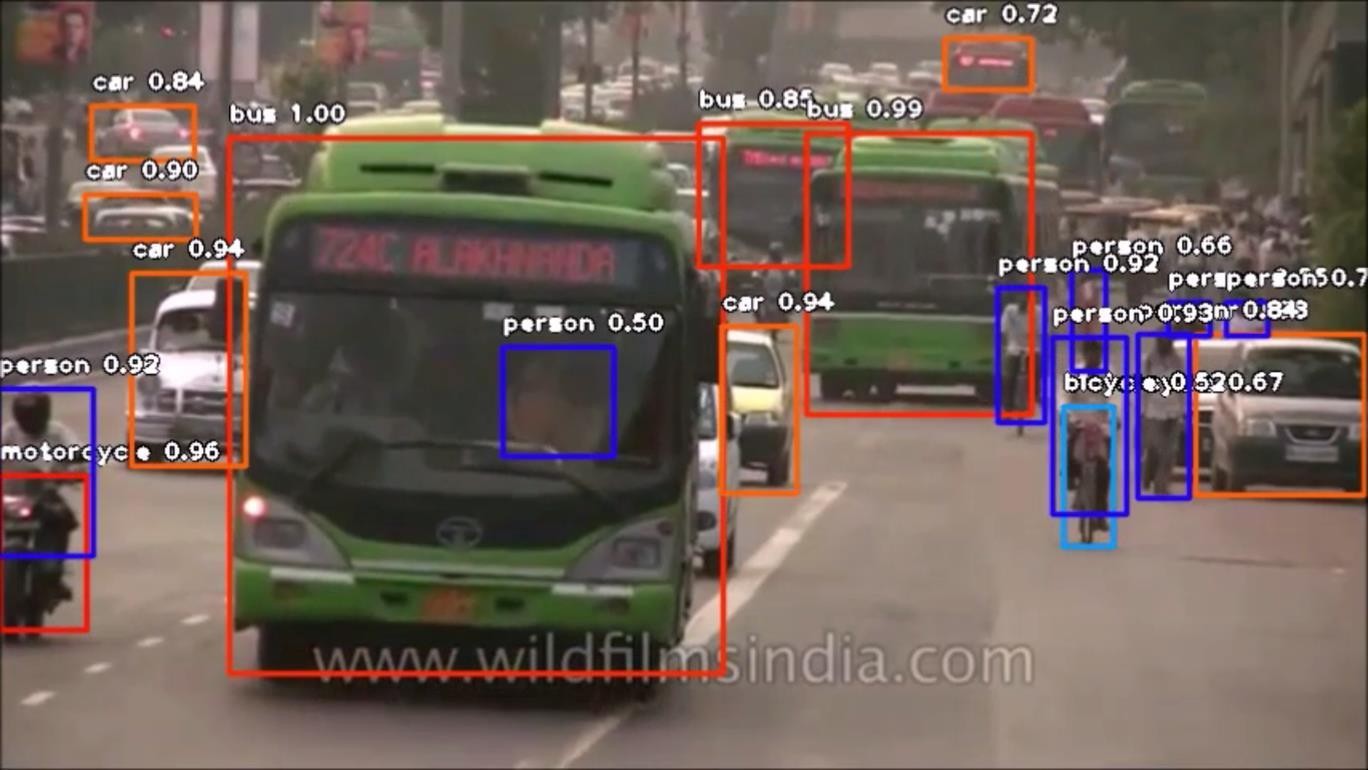
### Introduction

Object detection is technologically challenging and practically useful problem in the field of computer vision. Object detection deals with identifying the presence of various individual objects in an image. Great success has been achieved in controlled environment for object detection/recognition problem but the problem remains unsolved in uncontrolled places, in particular, when objects are placed in arbitrary poses in cluttered and occluded environment. As an example, it might be easy to train a domestic help robot to recognize the presence of coffee machine with nothing else in the image.

On the other hand imagine the difficulty of such robot in detecting the machine on a kitchen slab that is cluttered by other utensils, gadgets, tools, etc. The searching or recognition process in such scenario is very difficult. So far, no effective solution has been found for this problem. A lot of research is being done in the area of object recognition and detection during the last two decades. The research on object detection is multi-disciplinary and often involves the fields of image processing, machine learning, linear algebra, topology, statistics/probability, optimization, etc. The research innovations in this field

have become so diverse that getting a primary first hand summary of most state-of-the-art approaches is quite difficult and time consuming.

The approach used incorporates four computer vision and machine learning concepts: sliding windows to extract sub-images from the image, feature extraction to get meaningful data from the sub-images, Support Vector Machines (SVMs) to classify the objects in sub- image, and Principle Component Analysis (PCA) to improve efficiency. As a model problem for the motivating application, we focused on the problem of recognizing objects in images, in particular, soccer balls and sunflowers. For this algorithm to be useful as a real-time aid to the visually-impaired, it would have to be enhanced to distinguish between “close” and “far” objects, as well as provide information about relative distance between the user and the object, etc. We do not consider these complications in this project; we focus on the core machine learning issues of object recognition. The training and testing of the proposed algorithm was done using data sets .



Detecting objects Fig. 1.2.1

### Applications

* + 1. **Facial Recognition**



Fig. 1.3.1

A deep learning facial recognition system called the “**DeepFace**” has been developed by a group of researchers in the **Facebook,** which identifies human faces in a digital image very effectively. **Google** uses its own facial recognition system in Google Photos, which automatically segregates all the photos based on the person in the image. There are various components involved in Facial Recognition like the eyes, nose, mouth and the eyebrows.

* + 1. **People Counting**



Fig.1.3.2

Object detection can be also used for people counting, it is used for analyzing store performance or **crowd statistics** during festivals. These tend to be more difficult as people move out of the frame quickly.

* + 1. **Industrial Quality Check**



Fig. 1.3.3

Object detection is also used in industrial processes to identify products. Finding a specific object through visual inspection is a basic task that is involved in multiple industrial processes like sorting, inventory management, machining, quality management, packaging etc.

Inventory management can be very tricky as items are hard to track in **real time.** Automatic object counting and localization allows improving inventory accuracy.

* + 1. **Self Driving Cars**



Fig. 1.3.4

Self-driving cars are the Future, there’s no doubt in that. But the working behind it is very tricky as it combines a variety of techniques to perceive their surroundings, including radar, laser light, GPS, and computer vision.

### Purpose and need

The purpose of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image etc. In the case of a fixed rigid object only one example may be needed, but more generally multiple training examples are necessary to capture certain aspects of class variability.

One of the best examples of why you need object detection is the high-level algorithm for autonomous driving:

* In order for a car to decide what to do next: accelerate, apply brakes or turn, it needs to know where all the objects are around the car and what those objects are
* That requires object detection
* You would essentially train the car to detect known set of objects: cars, pedestrians, traffic lights, road signs, bicycles, motorcycles, etc.

### Hardware Specifications

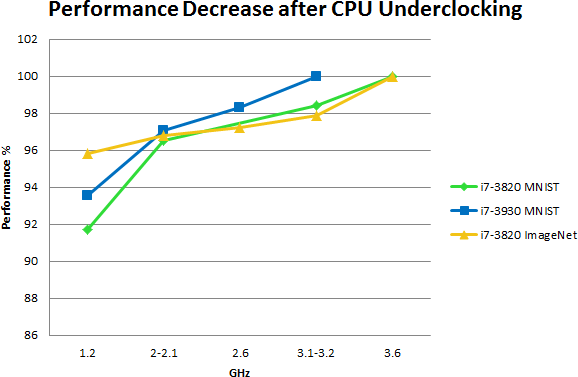
GPU For good cost/performance, I generally recommend an RTX 2070 or an RTX 2080 Ti. If you use these cards you should use 16-bit models. Otherwise, GTX 1070, GTX 1080, GTX 1070 Ti, and GTX 1080 Ti from eBay are fair choices and you can use these GPUs with 32-bit (but not 16-bit). Be careful about the memory requirements when you pick your GPU. RTX cards, which can run in 16-bits, can train models which are twice as big with the same memory compared to GTX cards. As such RTX cards have a memory advantage and picking RTX cards and learn how to use 16-bit models effectively will carry you a long way. In general, the requirements for memory are roughly the following:



### CPU

Suspect line-up Fig.2.4.1

The main mistake that people make is that people pay too much attention to PCIe lanes of a CPU. You should not care much about PCIe lanes. Instead, just look up if your CPU and motherboard combination supports the number of GPUs that you want to run.We need CPU with heavy RAM for running the large number of training steps



Performance Graph Fig.2.4.2

### Software Specifications

* + - Python 3.7
    - TensorFlow
    - Anaconda Software
    - Machine Learning Libraries

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language .

### Expected Outcome

Detection accuracy is usually measured on a given test set where the expected outcome for a detection sample is compared to the actual outcome of the object detection system .The detection accuracy is the percentage of samples for which the expected outcome matches the actual outcome of the detection system



Expected outcome Fig.2.6

# CHAPTER 2 BACKGROUND

### Machine learning

Learning algorithms are widely used in computer vision applications. Before considering image related tasks, we are going to have a brief look at basics of machine learning.

Machine learning has emerged as a useful tool for modelling problems that are otherwise difficult to formulate exactly. Classical computer programs are explicitly programmed by hand to perform a task. With machine learning, some portion of the human contribution is replaced by a learning algorithm. As availability of computational capacity and data has increased, machine learning has become more and more practical over the years, to the point of being almost ubiquitous.

##### Types

A typical way of using machine learning is supervised learning. A learning algorithm is shown multiple examples that have been annotated or labelled by humans. For example, in the object detection problem we use training images where humans have marked the locations and classes of relevant objects. After learning from the examples, the algorithm is able to predict the annotations or labels of previously unseen data. Classification and regression are the most important task type. In classification, the algorithm attempts to predict the correct class of a new piece of data based on the training data. In regression, instead of discrete classes, the algorithm tries to predict a continuous output.

In unsupervised learning, the algorithm attempts to learn useful proper- ties of the data without a human teacher telling what the correct output should be. Classical example of unsupervised learning is clustering. More recently, especially with the advent of deep learning technologies, un- supervised pre-processing has become a popular tool in supervised learning tasks for discovering useful representations of the data [9].

##### Features

Some kind of pre-processing is almost always needed. Pre-processing the data into a new, simpler variable space is called feature extraction. Of- ten, it is impractical or impossible to use the full-dimensional training data directly. Rather, detectors are programmed to extract interesting features from the data, and these features are used as input to the machine learning algorithm.

In the past, the feature detectors were often hand-crafted. The problem with this approach is that we do not always know in advance, which features are interesting. The trend in machine learning has been towards learning the feature detectors as well, which enables using the complete data.

##### Generalization

Since the training data cannot include every possible instance of the inputs, the learning algorithm has to be able to generalize in order to handle unseen data points. Too simple model estimate can fail to capture important aspects of the true model. On the other hand, too complex methods can overfit by modelling unimportant details and noise, which also leads to bad generalization. Typically, overfitting happens when a complex method is used in conjunction with too little training data. An overfitted model learns to model the known examples but does not understand what connects them.

The performance of the algorithm can be evaluated from the quality and quantity of errors. A loss function, such as mean squared error, is used to assign a cost to the errors. The objective in the training phase is to minimize this loss.

* 1. **Neural networks**

Neural networks are a popular type of machine learning model. A special case of a neural network called the convolutional neural network (CNN) is the primary focus of this thesis. Before discussing CNNs, we will discuss how regular neural networks work.

##### Origins

Neural networks were originally called artificial neural networks, because they were developed to mimic the neural function of the human brain. Pioneering research includes

the threshold logic unit by Warren McCulloch and Walter Pitts in 1943 and the perceptron by Frank Rosenblatt in 1957.

Even though the inspiration from biology is apparent, it would be mis- leading to overemphasize the connection between artificial neurons and biological neurons or neuroscience. The human brain contains approximately 100 billion neurons operating in parallel. Artificial neurons are mathematical functions implemented on more-or-less serial computers. Research into neural networks is mostly guided by developments in engineering and mathematics rather than biology.

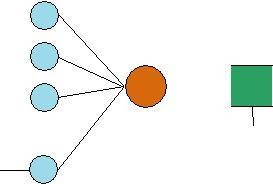


Figure 2.1: An artificial neuron.

An artificial neuron based on the McCulloch-Pitts model is shown in Figure. The neuron k receives m input parameters xj. The neuron also has m weight parameters wkj. The weight parameters often include a bias term that has a matching dummy input with axed value of 1. The inputs and weights are linearly combined and summed. The sum is then fed to an activation function ’ that produces the output yk of the neuron:

The neuron is trained by carefully selecting the weights to produce a desired output for each input.

##### Multi-layer networks

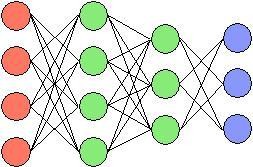


Figure 2.2: A fully-connected multi-layer neural network.

A neural network is a combination of artificial neurons. The neurons are typically grouped into layers. In a fully-connected feed-forward multi-layer network, shown in Figure 2.2 each output of a layer of neurons is fed as input to each neuron of the next layer. Thus, some layers process the original input data, while some process data received from other neurons. Each neuron has a number of weights equal to the number of neurons in the previous layer.

A multi-layer network typically includes three types of layers: an input layer, one or more hidden layers and an output layer. The input layer usually merely passes data along without modifying it. Most of the computation happens in the hidden layers. The output layer converts the hidden layer activations to an output, such as a classification. A multilayer feed-forward network with at least one hidden layer can function as a universal approximator i.e. can be constructed to compute almost any function.

In this thesis, we will mostly discuss fully-connected networks and convolutional networks. Convolutional networks utilize parameter sharing and have limited connections compared to fully-connected networks. Other network types, such as recurrent networks, are outside the scope of this thesis.

##### Back-propagation

A neural network is trained by selecting the weights of all neurons so that the network learns to approximate target outputs from known inputs. It is difficult to solve the neuron weights of a multi-layer network analytically. The back-propagation algorithm provides a

simple and effective solution to solving the weights iteratively. The classical version uses gradient descent as optimization method. Gradient descent can be quite time-consuming and is not guaranteed to find the global minimum of error, but with proper configuration (known in machine learning as hyper- parameters) works well enough in practice.

In the first phase of the algorithm, an input vector is propagated forward through the neural network. Before this, the weights of the network neurons have been initialized to some values, for example small random values. The received output of the network is compared to the desired output (which should be known for the training examples) using a loss function. The gradient of the loss function is then computed. This gradient is also called the error value. When using mean squared error as the loss function, the output layer error value is simply the difference between the current and desired output.

The error values are then propagated back through the network to calculate the error values of the hidden layer neurons. The hidden neuron loss function gradients can be solved using the chain rule of derivatives. Finally, the neuron weights are updated by calculating the gradient of the weights and subtracting a proportion of the gradient from the weights. This ratio is called the learning rate. The learning rate can be fixed or dynamic. After the weights have been updated, the algorithm continues by executing the phases again with different input until the weights converge.

In the above description, we have described online learning that calculates the weight updates after each new input. Online learning can lead to \zig-zagging" behavior, where the single data point estimate of the gradient keeps changing direction and does not approach the minimum directly. Another way of computing the updates is full batch learning, where we compute the weight updates for the complete dataset. This is quite computationally heavy and has other drawbacks. A compromise version is mini-batch learning, where we use only some portion of the training set for each update.

Mathematical descriptions of the algorithm are readily available in other works.

##### Activation function types

The activation function ’ determines the final output of each neuron. It is important to select the function properly in order to create an effective network.

Early researchers found that perceptron’s and other linear systems had severe drawbacks, being unable to solve problems that were not linearly separable, such as the XOR- problem. Sometimes, linear systems can solve these kinds of problems using hand-crafted feature detectors, but this is not the most advantageous use of machine learning. Simply adding layers does not help either, because a network composed of linear neurons remains linear no matter how many layers it has.

A light-weight and effective way of creating a non-linear network is using rectified linear units (ReLu). A rectified linear function generates the output using a ramp function such as:

This type of function is easy to compute and differentiate (for back- propagation). The function is not differentiable at zero, but this has not prevented its use in practice. ReLus have become quite popular lately, often replacing sigmoidal activation functions, which have smooth derivatives but suffer from gradient saturation problems and slower computation.

For multi-class classification problems, the softmax activation function is used in the output layer of the network:

The softmax function takes a vector of K arbitrarily large values and outputs a vector of K values that range between 0...1 and sum to 1. The values output by the softmax unit can be utilized as class probabilities.

##### Deep learning

Modern neural networks are often called deep neural networks. Even though multi-layer neural networks have existed since the 1980s, several reasons pre- vented the effective training of networks with multiple hidden layers.

One of the main problems is the curse of dimensionality. As the number of variables increases, the number of different configurations of the variables grows exponentially. As

the number of configurations increases, the number of training samples should increase in equal measure. Collecting a training dataset of sufficient size is time-consuming and costly or outright impossible.

Fortunately, real-world data is not uniformly distributed and often involves a structure, where the interesting information lies on a low-dimensional manifold. The manifold hypothesis assumes that most data configurations are invalid or rare. We can decrease dimensionality by learning to represent the data using the coordinates of the manifold. Another way to improve generalization is to assume local constancy. This means assuming that the function that the neural network learns to approximate should not change much within a small region.

In the past ten years, neural networks have had a renaissance, mainly because of the availability of more powerful computers and larger datasets. In early 2000s, it was discovered that neural networks could be trained efficiently using graphics processing units. GPUs are more efficient for the task than traditional CPUs and provide a relatively cheap alternative to specialist hardware. Today, researchers typically use high-end consumer graphic cards, such as NVIDIA Tesla K40.

Other more theoretical breakthroughs include replacing mean-squared error functions with cross-entropy based functions and replacing sigmoidal activation functions with rectified linear units.

With deep learning, there is less need for hand-tuned machine learning solutions that were used previously. A classical pattern detection system, for example, includes a hand- tuned feature detection phase before a machine learning phase. The deep learning equivalent consists of a single neural network. The lower layers of the neural network learn to recognize the basic features, which are then fed forward to higher layers of the network.

### Computer vision

Next, we are going to discuss computer vision in general and explore the primary subject of this thesis, object detection, as a subproblem of computer vision.

##### Overview

Computer vision deals with the extraction of meaningful information from the contents of digital images or video. This is distinct from mere image processing, which involves manipulating visual information on the pixel level. Applications of computer vision

include image classification, visual detection, 3D scene reconstruction from 2D images, image retrieval, augmented reality, machine vision and traffic automation.

Today, machine learning is a necessary component of many computer vision algorithms. Such algorithms can be described as a combination of image processing and machine learning. Effective solutions require algorithms that can cope with the vast amount of information contained in visual images, and critically for many applications, can carry out the computation in real time.

##### Object detection

Object detection is one of the classical problems of computer vision and is often described as a difficult task. In many respects, it is similar to other computer vision tasks, because it involves creating a solution that is invariant to deformation and changes in lighting and viewpoint. What makes object detection a distinct problem is that it involves both locating and classifying regions of an image. The locating part is not needed in, for example, whole image classification.

To detect an object, we need to have some idea where the object might be and how the image is segmented. This creates a type of chicken-and-egg problem, where, to recognize the shape (and class) of an object, we need to know its location, and to recognize the location of an object, we need to know its shape. Some visually dissimilar features, such as the clothes and face of a human being, may be parts of the same object, but it is difficult to know this without recognizing the object first. On the other hand, some objects stand out only slightly from the background, requiring separation before recognition.

Low-level visual features of an image, such as a saliency map, may be used as a guide for locating candidate objects. The location and size is typically defined using a bounding box, which is stored in the form of corner coordinates. Using a rectangle is simpler than using an arbitrarily shaped polygon, and many operations, such as convolution, are performed on rectangles in any case. The sub-image contained in the bounding box is

then classified by an algorithm that has been trained using machine learning. The boundaries of the object can be further refined iteratively, after making an initial guess.

During the 2000s, popular solutions for object detection utilized feature descriptors, such as scale-invariant feature transform (SIFT) developed by David Lowe in 1999 and histogram of oriented gradients (HOG) popularized in 2005. In the 2010s, there has been a shift towards utilizing convolutional neural networks.

Before the widescale adoption of CNNs, there were two competing solutions for generating bounding boxes. In the first solution, a dense set of region proposals is generated and then most of these are rejected. This typically involves a sliding window detector. In the second solution, a sparse set of bounding boxes is generated using a region proposal method, such as Selective Search. Combining sparse region proposals with convolutional neural networks has provided good results and is currently popular.

### Convolutional neural networks

Next, we are going to discuss why and how convolutional neural networks (CNN) are used and describe their history.

##### Justification

The problem with solving computer vision problems using traditional neural networks is that even a modestly sized image contains an enormous amount of information (see section 2.2.5 on deep learning and the curse of dimensionality).

A monochrome 620x480 image contains 297 600 pixels. If each pixel intensity of this image is input separately to a fully-connected network, each neuron requires 297 600 weights. A 1920x1080 full HD image would require 2,073,600 weights. If the images are polychrome, the amount of weights is multiplied by the amount of color channels (typically three). Thus, we can see that the overall number of free parameters in the network quickly becomes extremely large as the image size increases. Too large models cause over-fitting and slow performance.

Furthermore, many pattern detection tasks require that the solution is translation invariant. It is inefficient to train neurons to separately recognize the same pattern in the left-top corner and in the right-bottom corner of an image. A fully-connected neural network fails to take this kind of structure into account.

##### Basic structure

The basic idea of the CNN was inspired by a concept in biology called the receptive field. Receptive fields are a feature of the animal visual cortex. They act as detectors that are sensitive to certain types of stimulus, for example, edges. They are found across the visual field and overlap each other.

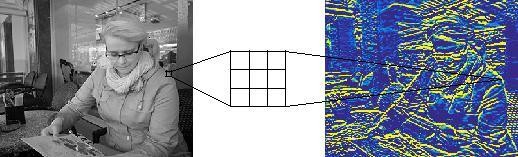


Figure 2.3: Detecting horizontal edges from an image using convolution filtering.

This biological function can be approximated in computers using the convolution operation. In image processing, images can be filtered using convolution to produce different visible effects. Figure 2.3 shows how a hand-selected convolutional filter detects horizontal edges from an image, functioning similarly to a receptive field.

The discrete convolution operation between an image f and a filter matrix g is defined as:

In effect, the dot product of the filter g and a sub-image of f (with same dimensions as g) centered on coordinates x; y produces the pixel value of h at coordinates x; y. The size of the receptive field is adjusted by the size of the filter matrix. Aligning the filter successively with every sub-image of f produces the of output pixel matrix h. In the case of neural networks, the output matrix is also called an feature map (or an activation map

after computing the activation function). Edges need to be treated as a special case. If image f is not padded, the output size decreases slightly with every convolution.

A set of convolutional filters can be combined to form a convolutional layer of a neural network. The matrix values of the filters are treated as neuron parameters and trained using machine learning. The convolution operation replaces the multiplication operation of a regular neural network layer. Output of the layer is usually described as a volume. The height and width of the volume depend on the dimensions of the activation map. The depth of the volume depends on the number of filters.

Since the same filters are used for all parts of the image, the number of free parameters is reduced drastically compared to a fully-connected neural layer. The neurons of the convolutional layer mostly share the same parameters and are only connected to a local region of the input. Parameter sharing resulting from convolution ensures translation invariance. An alternative way of describing the convolutional layer is to imagine a fully- connected layer with an infinitely strong prior placed on its weights. This prior forces the neurons to share weights at different spatial locations and to have zero weight outside the receptive field.

Successive convolutional layers (often combined with other types of layers, such as pooling described below) form a convolutional neural network (CNN). An example of a convolutional network is shown in figure. The back- propagation training algorithm, described in section 2.2.3 is also applicable to convolutional networks. In theory, the layers closer to the input should learn to recognize low-level features of the image, such as edges and corners, and the layers closer to the output should learn to combine these features to recognize more meaningful shapes. In this thesis, we are interested in studying whether convolutional networks can learn to recognize complete objects.

##### Pooling and stride

To make the network more manageable for classification, it is useful to de- crease the activation map size in the deep end of the network. Generally, the deep layers of the network require less information about exact spatial locations of features, but require

more filter matrixes to recognize multiple high-level patterns. By reducing the height and width of the data volume, we can increase the depth of the data volume and keep the computation time at a reasonable level.

There are two ways of reducing the data volume size. One way is to include a pooling layer after a convolutional layer. The layer effectively down-samples the activation maps. Pooling has the added effect of making the resulting network more translation invariant by forcing the detectors to be less precise. However, pooling can destroy information about spatial relationships between subparts of patterns. Typical pooling method is max- pooling. Max-pooling simply outputs the maximum value within a rectangular neighborhood of the activation map.

Another way of reducing the data volume size is adjusting the stride parameter of the convolution operation. The stride parameter controls whether the convolution output is calculated for a neighborhood centered on every pixel of the input image (stride 1) or for every nth pixel (stride n). Research has shown that pooling layers can often be discarded without loss in accuracy by using convolutional layers with larger stride value. The stride operation is equivalent to using a fixed grid for pooling.

##### Additional layers

The convolutional layer typically includes a non-linear activation function, such as a rectified linear activation function (see subsection 2.2.4). Activations are sometimes described as a separate layer between the convolutional layer and the pooling layer.

Some systems, such as also implement a layer called local response normalization, which is used as a regularization technique. Local response normalization mimics a function of biological neurons called lateral inhibition, which causes excited neurons to decrease the activity of neighboring neurons. However, other regularization techniques are currently more popular and these are discussed in the next section.

The final hidden layers of a CNN are typically fully-connected layers. A fully-connected layer can capture some interesting relationships parameter-sharing convolutional layers

cannot. However, a fully- connected layer requires a sufficiently small data volume size in order to be practical. Pooling and stride settings can be used to reduce the size of the data volume that reaches the fully-connected layers. A convolutional network that does not include any fully-connected layers, is called a fully convolutional network (FCN).

If the network is used for classification, it usually includes a softmax output layer (see also section 2.2.4) The activations of the topmost layers can also be used directly to generate a feature representation of an image. This means that the convolutional network is used as a large feature detector.

##### Regularization and data augmentation

Regularization refers to methods that are used to reduce overfitting by introducing additional constraints or information to the machine learning system. A classical way of using regularization in neural networks is adding a penalty term to the objective/loss function that penalizes certain types of weights. The parameter sharing feature of convolutional networks is another example of regularization.

There are several regularization techniques that are specific to deep neural networks. A popular technique called dropout attempts to reduce the co-adaptation of neurons. This is achieved by randomly dropping out neurons during training, meaning that a slightly different neural network is used for each training sample or minibatch. This causes the system not to depend too much on any single neuron or connection and provides an effective yet computationally inexpensive way of implementing regularization. In convolutional networks, dropout is typically used in the final fully-connected layers.

Overfitting can also be reduced by increasing the amount of training data. When it is not possible to acquire more actual samples, data augmentation is used to generate more samples from the existing data. For classification using convolutional networks, this can be achieved by computing transformations of the input images that do not alter the perceived object classes, yet provide additional challenge to the system. The images can be, for example, flipped, rotated or subsampled with different crops and scales. Also, noise can be added to the input images.

##### Development

Convolutional neural networks were one of the first successful deep neural networks. The Noncognition, developed by Fukushima in 1980s, provided a neural network model for translation-invariant object recognition, inspired by biology. Le Cun et al. combined this method with a learning algorithm, i.e. back-propagation. These early solutions were mostly used for hand- written character recognition.

After providing some promising results, the neural network methods faded in prominence and were mostly replaced by support vector machines. Then, in 2012, Krizhevsky et al. achieved excellent results on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset by com- bining Le Cun’s method with recent fine-tuning methods for deep learning. These results popularized CNNs and led to the development of new powerful object detection methods described in chapter.

For the 2014 ImageNet challenge, Simonyan and Zisserman explored the effect of increasing the depth of a convolutional network on localization and classification accuracy. The team achieved results that improved the then state-of-the-art by using convolutional networks 16 and 19 layers deep. The 16-layer architecture includes 13 convolutional layers (with 3x3 filters), 5 pooling layers (2x2 neighborhood max-pooling) and 3 fully-connected layers. All hidden layers use rectified (ReLu) activations. The fully-connected layers reduce 4096 channels down to 1000 softmax outputs and are regularized using dropout. This form of network is referred to as VGG-16 later in this thesis.

The current (2016) winner of the object detection category in the ImageNet challenge is also CNN-based. The method uses a combination of CRAFT region proposal generation, gated bi-directional CNN, clustering, landmark generation and ensembling.

# Chapter 3

**Convolutional Neural Networks**

#### R-CNN

In 2012, Krizhevsky et al. achieved promising results with CNNs for the general image classification task, as mentioned in section 2.4.6. In 2013, Girshick et al. published a method generalizing these results to object detection. This method is called R-CNN (\CNN with region proposals").

##### General description

R-CNN forward computation has several stages, shown in figure. First, the regions of interest are generated. The RoIs are category-independent bounding boxes that have a high likelihood of containing an interesting object. In the paper, a separate method called Selective Search, is used for generating these, but other region generation methods can be used instead. Selective Search, along with other region proposal generation techniques, is discussed in further detail in section 3.3.

Next, a convolutional network is used to extract features from each region proposal. The sub-image contained in the bounding-box is warped to match the input size of the CNN and then fed to the network. After the network has extracted features from the input, the features are input to support vector machines (SVM) that provide the final classification.



Figure 3.1: Stages of R-CNN forward computation.

The method is trained in multiple stages, beginning with the convolutional network. After the CNN has been trained, the SVMs are fitted to the CNN features. Finally, the region proposal generating method is trained.

##### Drawbacks

R-CNN is an important method, because it provided the first practical solution for object detection using CNNs. Being the first, it has many drawbacks that have been improved upon by later methods.

In his 2015 paper for Fast R-CNN, Girshick lists three main problems of R-CNN:

First, training consists of multiple stages, as described above. Second, training is expensive. For both SVM and region proposal training, features are extracted from each region proposal and stored on disk. This requires days of computation and hundreds of gigabytes of storage space.

Third, and perhaps most important, object detection is slow, requiring almost a minute for each image, even on a GPU. This is because the CNN forward computation is performed separately for every object proposal, even if the proposals originate from the same image or overlap each other.

#### Fast R-CNN

Fast R-CNN published in 2015 by Girshick provides a more practical method for object recognition. The main idea is to perform the forward pass of the CNN for the entire image, instead of performing it separately for each RoI.

##### General description

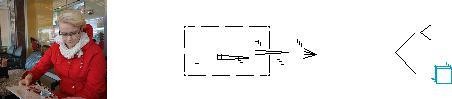


Figure 3.2: Stages of Fast R-CNN forward computation.

The general structure of Fast R-CNN is illustrated in figure 3.2. The method receives as input an image plus regions of interest computed from the image. As in R-CNN, the RoIs are generated using an external method. The image is processed using a CNN that includes several convolutional and max pooling layers.

The convolutional feature map that is generated after these layers is input to a RoI pooling layer. This extracts a fixed-length feature vector for each RoI from the feature map. The feature vectors are then input to fully- connected layers that are connected to two output layers: a softmax layer that produces probability estimates for the object classes and a real-valued layer that outputs bounding box co-ordinates computed using regression (meaning refinements to the initial candidate boxes).

##### Classification performance

According to the authors, Fast R-CNN provides significantly shorter classification time compared to regular R-CNN, taking less than a second on a state-of-the-art GPU. This is mainly due to using the same feature map for each RoI.

As the detection time decreases, the overall computation time begins to depend significantly on the performance of the region proposal generation method. The RoI generation can thus form a computational bottleneck. Additionally, when there are many RoIs, the time spent on evaluating the fully-connected layers can dominate the evaluation time of the convolutional layers. Classification time can be accelerated by approximately 30% if the fully-connected layers are compressed using truncated singular value decom- position. This results in a slight decrease in precision, however.

##### Training

According to the original publication, Fast R-CNN is more efficient to train than R-CNN, with nine-fold reduction in training time. The entire network (including the RoI pooling layer and the fully-connected layers) can be trained using the back-propagation algorithm and stochastic gradient de- scent. Typically, a pre-trained network is used as a starting point and then ne-tuned. Training is done in mini-batches of N images. R=N RoIs are sampled from each mini-batch image. The RoI samples are assigned to a class, if their intersection over union (see section 4.6) with a ground-truth box is over 0.5. Other RoIs belong to the background class.

As in classification, RoIs from the same image share computation and memory usage. For data augmentation, the original image is flipped horizontally with probability 0.5. The softmax classier and the bounding box regressors are ne-tuned together using a multi-task loss function, which con- siders both the true class of the sampled RoI and the o set of the sampled bounding box from the true bounding box.

#### Region proposal generation and use

To use R-CNN and Fast R-CNN, we need a method for generating the class-agnostic regions of interest. Next, we are going to discuss general principles of RoI generation, and have a closer look at two popular methods: Selective Search and Edge Boxes.

##### Overview

The aim of region proposal generation in object detection is to maximize recall i.e. to generate enough regions so that all true objects are recovered. The generator is less concerned with precision, since it is the task of the object detector to identify correct regions from the output of the region proposal generator.

However, the amount of proposals generated affects performance. As mentioned in section 2.3.2 there are two main approaches to region generation: dense set generation and sparse set generation.

Dense set solutions attempt to generate by brute force an exhaustive set of bounding boxes that includes every potential object location. This can be achieved by sliding a

detection window across the image. However, searching through every location of the image is computationally costly and requires a fast object detector. Additionally, different window shapes and sizes need to be considered. Thus, most sliding window methods limit the amount of candidate objects by using a coarse step-size and a limited number of fixed aspect ratios.

Most region proposals in a dense set do not contain interesting objects. These proposals need to be discarded after the object detection phase. Detection results can be discarded, if they fall behind a predefined confidence threshold or if their confidence value is below a local maximum (non-maximum suppression).

Instead of discarding the regions after the object detection stage, the region proposal generator itself can rank the regions in a class-agnostic way and discard low-ranking regions. This generates a sparse set of object detections[.](https://www.htmlpublish.com/newTestDocStorage/DocStorage/35ae8f4a8ce34d87afe66d65e9fa45e7/pdf-to-word.htm#page_75) Similarly to dense set methods, thresholding and non-maximum suppression can be implemented after the detection phase to further improve the detection quality. Sparse set solutions can be grouped into unsupervised and supervised methods.

One of the most popular unsupervised methods is Selective Search (see section 3.3.2) which utilizes an iterative merging of superpixels. There are also other methods that use the same approach. Another approach is to rank the objectness of a sliding window. A popular example of this is Edge Boxes (see section 3.3.2) which calculates the objectness score by calculating the number of edges within a bounding box and by subtracting the number of edges that overlap the box boundary. There is also a third group of methods based on seed segmentation.

Supervised methods treat region proposal generation as a classification or a regression problem. This means using a machine learning algorithm, such as a support vector machine. It is also possible to use a convolutional network to generate the regions of interest. An example of using a CNN for calculating the bounding boxes is Multi-Box.

Certain advanced object detection methods, such as Faster R-CNN described in 3.4.1 use parts of the same convolutional network both for generating the region proposals and for detection. We call these kinds of methods integrated methods.

##### Selective Search

Selective Search utilizes a hierarchical partitioning of an image to create a sparse set of object locations. The main design philosophy is not to use a single strategy, but to combine the best features of bottom-up segmentation and exhaustive search. The authors had three main design considerations: the search should capture all scales, be diverse i.e. not use any single strategy for grouping regions and be fast to compute.

The algorithm begins by creating a set of small initial regions using a method called Graph Based Image Segmentation designed by Felzen- szwalb and Huttenlocher. The method creates a set of regions called super- pixels. The superpixels are internally nearly uniform. Combined, they span the entire image, but individually they should not span different objects.

Selective Search then continues by iteratively grouping the regions together using a greedy algorithm, beginning with the two most similar regions. Many complimentary measures are used to compute the similarity. These measures consider color similarity (by computing a color histogram), texture similarity (by computing a SIFT-like measure), size of the regions (small regions should be merged earlier) and how well the regions fit together (gaps should be avoided). The grouping phase ends when every region has been combined.

The hypothetical object locations thus generated are then ordered by the likelihood of the location containing an object. In practice, the locations are ordered based on the order in which they were grouped together by the different measures. A certain element of randomness is added to prevent large objects from being favored too much. Lower- ranking duplicates are removed.

Both the region generating method and the similarity measures were selected to be fast to compute, making the method fast in general. In addition to using diverse similarity measures, the search can be further diversified by using complementary color spaces (to ensure lighting invariance) and using complementary starting regions.

##### Edge Boxes

As the name suggests, Edge Boxes is based on detecting objects from edge maps. The main contribution of the authors of the method is the observation that the number of edge contours wholly enclosed by a bounding box is correlated with the likelihood that the box contains an object.

First, the edge map is calculated using a method by the same authors called Structured Edge Detector. Then, thick edge lines are thinned using non-maximum suppression.

Instead of operating on the edge pixels directly, the pixels are grouped using a greedy algorithm. An affinity measure is devised to calculate whether edge groups are part of the same contour.

The region proposals are found by scanning the image using the traditional sliding window method and calculating an objectness score at each position, aspect ratio and scale. The score is calculated by summing the edge strength of edge groups that lie completely within the box and subtracting the strength of edge groups that are part of a contour that cross the box boundary. Promising regions are then further refined.

#### Advanced convolutional object detection

In the experimental section of this thesis, we will focus mostly on Fast R- CNN. There are, however, several state-of-the-art algorithms with an im- proved computation time or accuracy. Next, we will describe two of these algorithms. See also chapter [7](https://www.htmlpublish.com/newTestDocStorage/DocStorage/35ae8f4a8ce34d87afe66d65e9fa45e7/pdf-to-word.htm#page_59) for discussion of improvements of convolutional object detection.

##### Faster R-CNN

Faster R-CNN by Ren et al. is an integrated method. The main idea is to use shared convolutional layers for region proposal generation and for detection. The authors discovered that feature maps generated by object detection networks can also be used to generate the region proposals. The fully convolutional part of the Faster R-CNN network that generates the feature proposals is called a region proposal network (RPN). The authors used Fast R-CNN architecture for the detection network.

A Faster R-CNN network is trained by alternating between training for RoI generation and detection. First, two separate networks are trained. Then, these networks are combined and fine-tuned. During fine-tuning, certain layers are kept fixed and certain layers are trained in turn.

The trained network receives a single image as input. The shared fully convolutional layers generate feature maps from the image. These feature maps are fed to the RPN. The RPN outputs region proposals, which are input, together with the said feature maps, to the final detection layers. These layers include a RoI pooling layer and output the final classifications.

Using shared convolutional layers, region proposals are computationally almost cost-free. Computing the region proposals on a CNN has the added benefit of being realizable on a GPU. Traditional RoI generation methods, such as Selective Search, are implemented using a CPU.

For dealing with different shapes and sizes of the detection window, the method uses special anchor boxes instead of using a pyramid of scaled images or a pyramid of different filter sizes (see section 7.2 for discussion of scale invariance). The anchor boxes function as reference points to different region proposals centered on the same pixel.

##### SSD

The Single Shot MultiBox Detector (SSD) takes integrated detection even further. The method does not generate proposals at all, nor does it involve any resampling of image segments. It generates object detections using a single pass of a convolutional network.

Somewhat resembling a sliding window method, the algorithm begins with a default set of bounding boxes. These include different aspect ratios and scales. The object predictions calculated for these boxes include o set parameters, which predict how much the correct bounding box surrounding the object identifiers from the default box.

The algorithm deals with different scales by using feature maps from many different convolutional layers (i.e. larger and smaller feature maps) as input to the classier. Since the method generates a dense set of bounding boxes, the classier is followed by a non- maximum suppression stage that eliminates most boxes below a certain confidence threshold.

#### Comparing the methods

Above, we described how Fast R-CNN is faster and more accurate than regular R-CNN. But how does Fast R-CNN perform compared to the above- mentioned advanced methods?

Liu et al. compared the performance of Fast R-CNN, Faster R-CNN and SSD on the PASCAL VOC 2007 test set (see section 4.5 for discussion of the standard benchmarks). When using networks trained on the PASCAL VOC 2007 training data, Fast R-CNN achieved a mean average precision (mAP) of 66.9 (see section 4.6 or discussion of evaluation methods). Faster R-CNN performed slightly better, with a mAP of 69.9. SSD achieved a mAP of 68.0 with input size 300 x 300 and 71.6 with input size 512 x 512. As the standard implementations of Fast R-CNN and Faster R-CNN use 600 as the length of the shorter dimension of the input image, SSD seems to perform better with similarly sized images. However, SSD requires extensive use of data augmentation to achieve this result. Fast R-CNN and Faster R- CNN only use horizontal flipping, and it is currently unknown, whether they would benefit from additional augmentation.

While the advanced methods are more precise than Fast R-CNN, the real improvements come from speed. When most of the detections with a low probability are eliminated using thresholding and non-maximum suppression (see section 4.6 for details), SSD512

can run at 19 FPS on a Titan X GPU. Meanwhile, Faster R-CNN with a VGG-16 architecture performs at 7

FPS. The original authors of Faster R-CNN report a running time of 5 FPS i.e. 0.2 s per image. Fast R-CNN has approximately the same evaluation speed, but requires additional time for calculating the region proposals. Region generation time depends on the method, with Selective Search re- quiring 2 seconds per image on a CPU and Edge Boxes requiring 0.2 seconds per image.

# CHAPTER 4 SYSTEM DESIGN

### Data Flow Diagrams

A representation of a system at different levels of details with graphic nexus of symbols representing data flows, data stores, procedures and data end points like source and destinations is termed as Data Flow Diagram.

### Design Notations

##### Process

A procedure or process does operations and give the output on the supplied arguments. The pure

Functions are considered as low level process that do not have side effects. A process data flow component is represented as an ellipse.

* + - **Data Flows**

The nexus between one process to another or one sub identity to mother is represented by the with the intermediate value or the label on it

Graphical Representation

##### Actors

The element that drives the data flow by taking the inputs and thereby computing the out is termed as the actor.

##### Data Store

Sometimes data is required to be accessed later in the data flow that is done by data store component of DFD.

##### External Entity

Any external entity which can access the flow in DFD like a librarian, is called as External Entity component. It is represented as a rectangle.

Graphical Representation

##### Output Symbol

While the user interaction with the system the DFD depicts it in the form of a below polygon.

Graphical Representation

### Detailed Design

**Zero level DFD – object identification system**

Object detection

Recognition

INPUT IMAGE

Fig. 4.3.1

### First Level DFD-

##### Pre-Processing image

Pre-Processing Image

10 Different Direction

28 Dimension

Convert it into gray image

Fig. 4.3.2

##### Processing

Pre-Processing Image

Find the neighborhood

Find the nodal points

Compare the Image

Testing the Image

Fig. 4.3.3

##### Recognition

Processing Image

Compare the both

Compare maximum

Testing Image

Retrived the stored Image

Fig. 4.3.4

##### Testing the Image

Testing the Image

Yes

No

Matched

Not Matched

### Second level DFD

Fig. 4.3.5



Capture the Image



Recognition

Processing

Pre-Processing

Convert it into gray Image

Pre-Processing Image

Pre-Processing Image

Processing Image

Convert gray image

Find the neighbourhood the nodal points

Retrieve the stored Image

10 different directions

Find the nodal points

Compare both image

Testing the image

Testing the Image

Compare the Image

28 Dimensions

Compare Maximal Percentage

Fig. 4.3.6

### Use case



Input gray

Pre-Processing

Processing

Recognition

Fig. 4.3.7

### Sequence Diagram

Give Extracted Gray

Level Of Pixel

Compare Calculated Principle Points with Test Image

If The Image Is Rec- Ognized, Then It Show To The user

Train Image

Processing

Pre-Processing

User

Input Gray Image

Fig.5.3.8

# Testing

Unit Testing

Module Testing

Sub-System

Testing

System Setting

Acceptance Testing

(Component testing)

A set of activities carried out to check a the functionality or stability is termed as testing. These activities are so planned and perfomed systematically that it leaves no scope for rework or bugs. General characteristics of this strategies are:

1. Testing begins at the module level and works outward".
2. disparate testing techniques are appropriate at disparate points in time.
3. Debugging and testing are altogether disparate procedures.
4. The developer of the software conducts testing and if the project is big then there is a testing team.

The System testing involved is the most widely used testing procedure consisting of five stages as shown in the figure. In general, the sequence of testing activities is component testing, integration testing, and then user testing

(Integration testing)

### Functional Testing

Fig.8.1

(User testing)

Once the system is completed developed and integrated it is checked and evaluated for its functionality as whole for specific demands and requirements. This type of testing falls under the category of Blackbox testing and does not require the knowledge of in depth working and protocol off the system.

### Structural Testing

In contrary to Functional testing Structural testing checks for the functionality of the different modules of the whole system and how well they are in link with other module. This type of testing requires full knowledge of the behaviour, protocol and the working of the system as a whole and module wise. The system base coding and programming knowledge is also a requirement to perform this testing. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

### Testing the model

To test the model, we first select a model checkpoint (usually the latest) and export into a frozen inference graph. checkpoints is created when we train our model with the help of checkpoint we are testing our model . we divide our data and used 70% images for training and 30% for testing purpose so we split our images in test and train train folder

We store 100 of images per object to train the model of every angle of the object . In figure

9.3 there are some test images

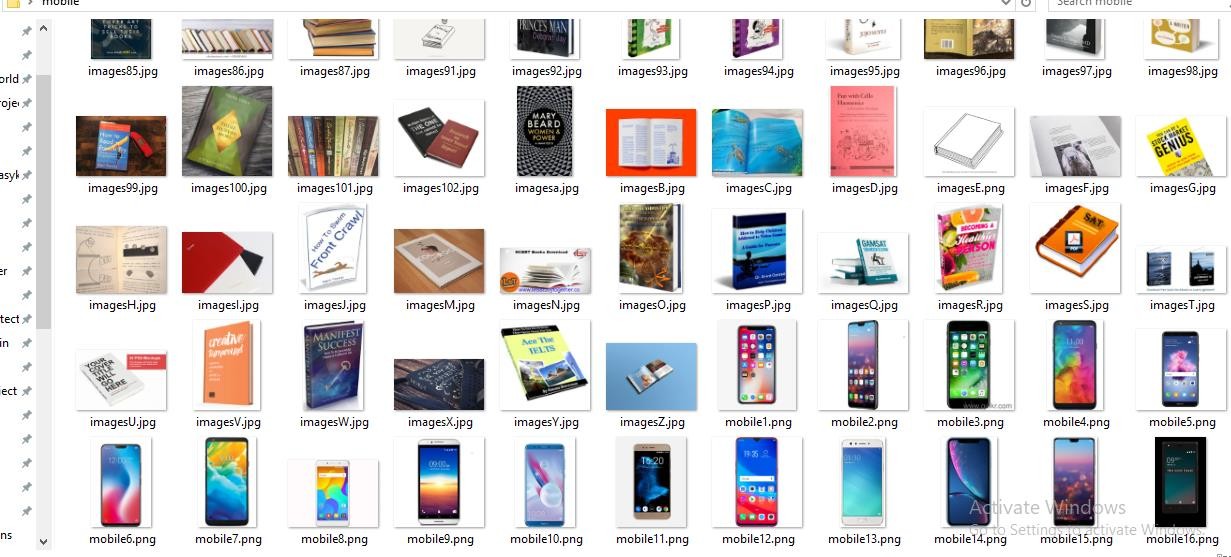


Fig 8.3.1

We ran tests with databases built for 6,12,18,24 objects and obtained overall success rates(correct classification on forced choice) of 99.6%, 98%, 97.4% and 97% respectively. The worst cases were the book and the pen in 24 object test,with 19/24 and 20/24 correct respectively

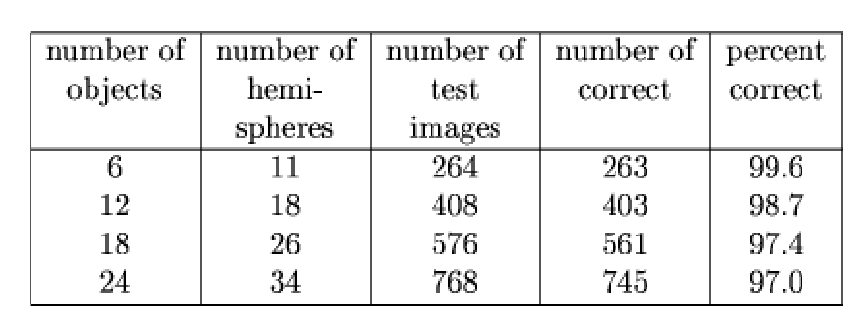


Table 8.3.2

The time to identify an object depends more or less linearly on the number of key features fed to the system, and the size of the database. At the moment, overall recognition time on a single processor are about 20 seconds for the 6 object database, and about 2 min for the 24 object database. This could also be improved substantially by pushing on the indexing methods.

The program updates the video window with a new frame every between 0.25 sec and 0.5 seconds, which means an average of 2 - 4 FPS. In this project we detect live object with help of camera.

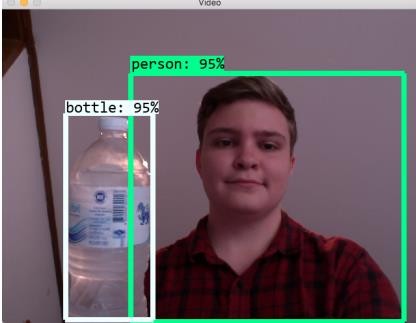


Fig. 8.3.3

It identifies me as a person with 95% confidence and water bottle also with 95% confidence. It show the accuracy of detecting the object



Fig. 8.3.4

**Chapter 9**

**Maintenance & Evaluation**

Maintenance is the is the term that is used to refer to modifications that are made to software system after its release. System maintenance is an ongoing activity which covers a wide variety of activities including removing program and design errors, updating documentation and test data and updating user support Maintenance can be broadly classified into following three classes:

### Corrective maintenance

This is used to remove errors in the program, which occurs when the product is delivered as well as during maintenance. Thus in corrective maintenance the product is modified to solve the discovered errors after the software product is being delivered to customer.

### Adaptive maintenance

Adaptive maintenance is generally not requested by client but it is imposed by the outside environment. It may include following organizational changes:

* + - Change in the object
    - Change in algorithms for faster performance
    - Change in frames like instead of live detecting we need video frames
    - Change in system controls and security needs etc.

### Perfective maintenance

It means changing the software to improve some of its qualities like add new functions improve computer efficiency, make it easier to use. This type of maintenance is used to

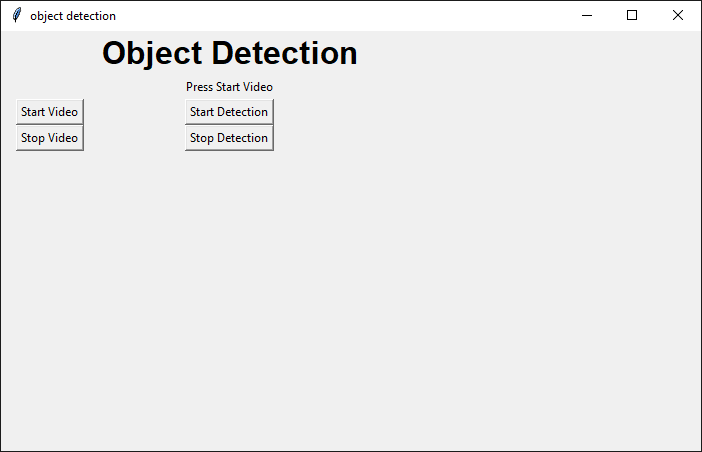
respond to user's additional needs may be due to the changes within or outside of the organization. These changes include:

* + - Changes in software
    - Economic and competitive conditions
    - Changes in models

System evaluation is the process of checking the performance of a complete system to acknowledge how it is likely to perform in live market conditions. It measures the performance of the system that whether it may compete or not.

# Chapter 10

**CODING AND SCREEN**



import tkinter as tk

import settings

import cv2

from PIL import Image, ImageTk

import numpy as np

import imutils

def start\_video():

settings.start\_video = True

show\_frame()

def stop\_video():

settings.start\_video = False

settings.start\_processing = False

lmain.config(image='')

def start\_process():

settings.start\_processing = True

def stop\_process():

settings.start\_processing = False

def show\_frame():

if not settings.start\_video:

return None

\_, frame = cap.read()

frame = cv2.flip(frame, 1)

frame = imutils.resize(frame, width=400)

if settings.start\_processing:

frame = process\_frame(frame)

cv2image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(cv2image)

imgtk = ImageTk.PhotoImage(image=img)

lmain.imgtk = imgtk

lmain.configure(image=imgtk)

lmain.after(10, show\_frame)

def process\_frame(img):

# grab the frame dimensions and convert it to a blob

(h, w) = img.shape[:2]

blob = cv2.dnn.blobFromImage(cv2.resize(img, (300, 300)),

0.007843, (300, 300), 127.5)

# pass the blob through the network and obtain the detections and

# predictions

net.setInput(blob)

detections = net.forward()

# loop over the detections

for i in np.arange(0, detections.shape[2]):

# extract the confidence (i.e., probability) associated with

# the prediction

confidence = detections[0, 0, i, 2]

# filter out weak detections by ensuring the `confidence` is

# greater than the minimum confidence

if confidence > 0.2:

# extract the index of the class label from the

# `detections`, then compute the (x, y)-coordinates of

# the bounding box for the object

idx = int(detections[0, 0, i, 1])

box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

(startX, startY, endX, endY) = box.astype("int")

# draw the prediction on the frame

label = "{}: {:.2f}%".format(CLASSES[idx],

confidence \* 100)

cv2.rectangle(img, (startX, startY), (endX, endY),

COLORS[idx], 2)

y = startY - 15 if startY - 15 > 15 else startY + 15

cv2.putText(img, label, (startX, y),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, COLORS[idx], 2)

return img

CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",

"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",

"dog", "horse", "motorbike", "person", "pottedplant", "sheep",

"sofa", "train", "tvmonitor"]

COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk

print("Loading model...")

net = cv2.dnn.readNetFromCaffe('MobileNetSSD\_deploy.prototxt.txt', 'MobileNetSSD\_deploy.caffemodel')

cap = cv2.VideoCapture(0)

window = tk.Tk()

window.title("object detection")

window.geometry('700x420')

lbl = tk.Label(window, text="Object Detection", font=("Arial Bold", 24))

lbl.grid(column=1, row=0)

imageFrame = tk.Frame(window, width=600, height=500)

imageFrame.grid(row=1, column=1, padx=10, pady=2)

lmain = tk.Label(imageFrame, text="Press Start Video")

lmain.grid(row=1, column=1)

startVideoStreamBtn = tk.Button(window, text="Start Video", command=start\_video)

startVideoStreamBtn.grid(column=0, row=2, padx=15)

stopVideoStreamBtn = tk.Button(window, text="Stop Video", command=stop\_video)

stopVideoStreamBtn.grid(column=0, row=3, padx=15)

startProcessBtn = tk.Button(window, text="Start Detection", command=start\_process)

startProcessBtn.grid(column=1, row=2)

stopProcessBtn = tk.Button(window, text="Stop Detection", command=stop\_process)

stopProcessBtn.grid(column=1, row=3)

window.mainloop()

import cv2

import argparse

import numpy as np

ap = argparse.ArgumentParser()

ap.add\_argument('-i', '--image', required=True,

help = 'path to input image')

ap.add\_argument('-c', '--config', required=True,

help = 'path to yolo config file')

ap.add\_argument('-w', '--weights', required=True,

help = 'path to yolo pre-trained weights')

ap.add\_argument('-cl', '--classes', required=True,

help = 'path to text file containing class names')

args = ap.parse\_args()

def get\_output\_layers(net):

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

return output\_layers

def draw\_prediction(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)

image = cv2.imread(args.image)

Width = image.shape[1]

Height = image.shape[0]

scale = 0.00392

classes = None

with open(args.classes, 'r') as f:

classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

net.setInput(blob)

outs = net.forward(get\_output\_layers(net))

class\_ids = []

confidences = []

boxes = []

conf\_threshold = 0.5

nms\_threshold = 0.4

for out in outs:

for detection in out:

scores = detection[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])

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\_prediction(image, class\_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))

cv2.imshow("object detection", image)

cv2.waitKey()

cv2.imwrite("object-detection.jpg", image)

cv2.destroyAllWindows()

# Chapter 11

**Conclusion and Future Scope**

The Object Detection system in Images is web based application which mainly aims to detect the multiple objects from various types of images. To achieve this goal shape and edge feature from image is extracted. It uses large image database for correct object detection and recognition. This system will provide easy user interface to retrieve the desired images. The system have additional feature such as Sketch based detection. In Sketch detection user can draw the sketch by hand as an input. Finally the system results output images by searching those images that user want.

### Scope of Object Detection and Recognition

The project has wide scope in multiple areas and can easily increase its utilization by adding more efficient algorithms. Some of the areas are as follows-

##### Medical Diagnose:

Use of object detection and recognition in medical diagnose to detect the X-Ray report, brain tumors.

##### Shapes recognition:

Recognize the shape from whole region in images.

##### Cartography:

The cartography as the discipline dealing with the conception, production dissemination and study of maps.

##### Robotics:

In robotics use of object detection is movement of body parts and motion sensing.

# Chapter 11 References

* + - He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2016), pp. 770–778.
    - Hoiem, D., Efros, A. A., and Hebert, M. Automatic photo popup. ACM transactions on graphics (TOG) 24, 3 (2005), 577–584.
    - Hoiem, D., Efros, A. A., and Hebert, M. Geometric context from a single image. In Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on (2005), vol. 1, IEEE, pp. 654–661.
    - Hoiem, D., Efros, A. A., and Hebert, M. Putting objects in perspective. International Journal of Computer Vision 80, 1 (2008), 3–15.
    - Hornik, K. Approximation capabilities of multilayer feedforward networks. Neural networks 4, 2 (1991), 251–257.
    - Huang, T. Computer vision: Evolution and promise. CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH-REPORTSCERN (1996), 21– 26.
    - Hubel, D. H., and Wiesel, T. N. Receptive ﬁelds and functional architecture of monkey striate cortex. The Journal of Physiology 195, 1 (1968), 215–243.
    - Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167 (2015).
    - Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long,
    - a