Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

A Comprehensive Classification and Evaluation of Deep Learning Techniques for Object Detection in Videos

Mihir Mulye

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
M. Sc. Deebul Nair
M. Sc. Santosh Thoduka

January 2020

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Mihir Mulye

Abstract

The performances of object detectors in images have improved drastically over the years but the same has not been replicated in videos due to problems like occlusions, blurring and inference time. However, the temporal aspect implicitly incorporated in the videos might provide a solution to handle these problems.

This research work discusses the approaches available for conducting object detection in videos and also attempts to segregate the existing methods into systematic categories. A pattern amongst the analysed methods was identified and three categories: temporal information usage, framewise feature extraction and adopted paradigm (approach) were established.

From the analysis of the methods, two approaches were selected to investigate further by implementing them, namely *Framewise* and *Coupled*. These methods aim to explore the role of temporal information in tackling scenarios like occlusions. Owing of the advancements in image object detection, framewise approach by extension can be used to detect objects in videos with higher accuracy albeit lower speeds. This provides a reference against which the accuracy of the other method can be compared. *Framewise* approach has been implemented by adapting pretrained SSD MobileNet architecture and it incorporates no temporal information. The *Coupled* approach makes use of the same SSD MobileNet with Open Loop Kalman Filter. A collection of videos were recorded for the purpose of testing the performance of the approaches on real world scenarios. These were recorded using USB camera and Mobile camera for introducing variation in the recordings. Based on the experiments, it was found that the *Coupled* approach performs better than *Framewise* approach for majority of the occlusion scenarios with the parameters of analysis being Intersection over Union and Inference Time. Even in the scenarios where *Framewise* approach performed better, it was marginally so.

Acknowledgements

I would like to take this opportunity to appreciate the efforts of people who have supported and helped me in this undertaking. First and foremost, I would express my sincere gratitude to my supervisors Prof. Dr. Paul G. Plöger, M. Sc. Deebul Nair and M. Sc. Santosh Thoduka for their guidance and timely inputs. I received continued encouragement and support from them throughout my project. The research work was crafted as a result of the numerous stimulating discussions found on the online community at stackexchange and GitHub which resulted in betterment of my understanding. Finally, I would like to thank my mother for providing me with this opportunity to pursue Master Studies and keep me motivated throughout this period. I would also like to take this opportunity to thank Ashay Mulye for reviewing my drafts numerous times and providing me with constructive feedback to improve. This research work has been an opportunity to learn and grow not only professionally but also personally and it would not have been possible without the mentoring and guidance from a lot of people.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

With the recent advances in technology, humans are developing systems that can operate independently. Such systems are referred to as *Autonomous Systems*. Examples of these systems are found in applications such as Auto-pilot in an aircraft, a robot vacuum cleaner, assembly line of a factory and so on. As research is progressing, the complexity and the capabilities of autonomous systems are increasing significantly. From the humble beginnings of hardwired path following robots, humans are now in the process of developing cars that can drive without help from humans. The transition from no autonomy to full autonomy is however not as discrete as it appears to be. The level of human involvement and operating conditions are some of the factors that determine the *degree of autonomy* of the system. Lesser the human involvement, higher the *degree of autonomy*. Autonomous Systems comprises of three components which are discussed in the following section.

1.1.1 Components of Autonomous Systems

Majority of the autonomous systems strive to mimic human behaviour when performing a task and follow the *sense, plan* and *act* cycle to do so[1]. This cycle is depicted in Figure 1.1. With reference to Figure 1.1, the components are explained as follows:

- **Sense** (also referred to as *Perception*): when an autonomous system has to perform a certain task, it collects data from the surroundings to gather information about the current state of the environment. These measurements can be in the form of visual, numeric or audio data. Depending on the type of data needed, sensors can be selected.

[1] http://www.gkmm.tu-darmstadt.de/files/RAS_Short_Course_OvS_Part1-bw.pdf

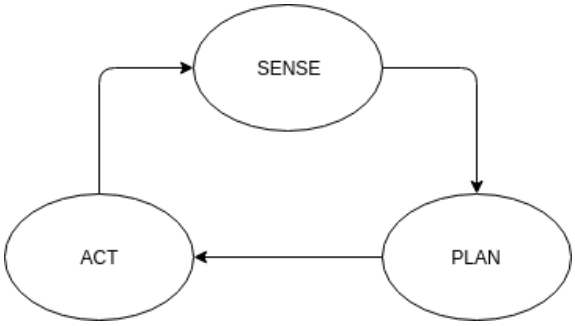


Figure 1.1: Sense, Plan and Act cycle[2]

- **Plan**: based on the measured state of the system a *plan* to achieve the desired state is formalized. The *plan* step can involve decision making process, trajectory planning process or calculating the sequence of actuator activation.

- **Act**: once the *plan* is established, the actuators of the system perform the task. Here, a feedback mechanism can be implemented which would determine the success of the task. All the three steps together constitute one iteration of the cycle.

To be aware of the surroundings, *sense* step needs to be continuous in nature rather than being discrete. This work exclusively deals with the *sense* (perception) step of the cycle, as it is critical and is in execution at all points of time. Sensing can be done in multiple ways with vision being one of the important ones. Recognizing the importance of perception, this work focuses on the domain of vision. Object detection, a sub-task of the computer vision field is discussed in this. Following section contains an overview of the tasks in vision domain.

1.2 Introduction to Vision Tasks

Many crucial tasks fall under the purview of computer vision, some of which are object classification, object detection and semantic segmentation. These tasks may sound simple for humans since we have a keen sense of vision. However, for robots and other autonomous systems, these can prove to be difficult. Hence, it is important to clearly define these tasks for visual systems. Following provides a brief overview of the tasks in vision systems:

- **Classification**
  Given an input image, the output of this task is the *class* of object contained in the image. This can be seen in Figure 1.2. Many deep learning architectures also provide the probability score of each class thus providing the confidence on the results as well.

[2] 1

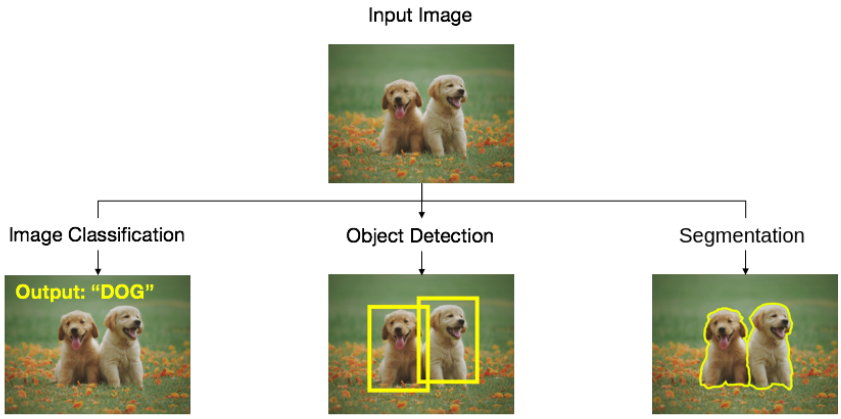


Figure 1.2: Different type of Vision Tasks[3]

- **Detection**
  Given an input image, the output is the *class* as well as the *location* of the detected object in terms of *bounding box* coordinates. Since this task is of interest here, a detailed explanation of it would be provided in the next section.

- **Segmentation**
  The key focus of this task is the categorization of each pixel in input image to corresponding categories. Instead of providing a rectangular region where the object lies (as in the case of detection), segmentation output provides the outline of the object and hence yields a precise localization of the object. Two sub categories of this task are: *semantic segmentation* and *instance segmentation*. *Instance segmentation* differs from *Semantic segmentation* in that it assigns a different color to objects of same class. The segmentation depicted in Figure 1.2 is *semantic segmentation*.

1.3 Object Detection

In the previous section, various vision tasks were introduced and the differences between them were discussed. This section deals with the task of detection and discusses it in detail. Object detection is a technique for identifying instances of objects and locating their position in images or in videos [8]. The task of object detection would be explained by taking an example from Coursera [1] where the discussions of [45] are lucidly explained. The subsequent explanation has been taken from the aforementioned sources. Figure 1.3 can be used as a reference to understand the following discussion.

As can be seen in Figure 1.3, the output of object detection can be understood in the form of a vector. The authors of [45] described an algorithm named as You Only Look Once (YOLO),

[3]Image adapted from: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789138900/4/ch04lvl1sec43/types-of-object-recognition-tasks

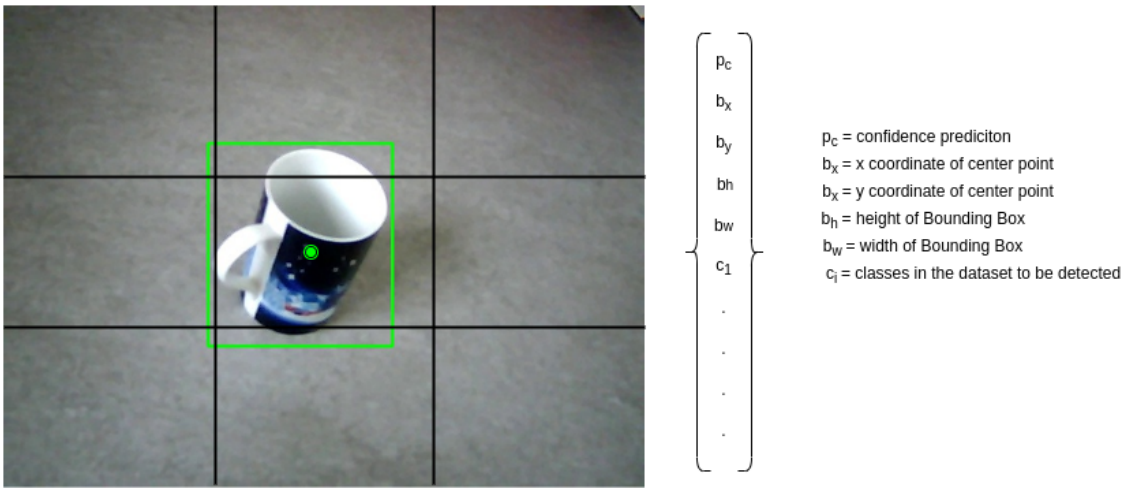


Figure 1.3: Object Detection output explained

to conduct object detection on images. As per the algorithm, the image is divided into grids as shown and then the algorithm is run on every grid cell. The authors argue that object is considered belonging to that grid cell where the mid point of object lies. In Figure 1.3, this would be the central cell. Though, the adjoining cells have some part of the object in them the object is still considered to be lying in the central cell. The vector provides the output of the algorithm for each grid cells. The number of classes to be detected determines the size of this vector. For instance if 3 object classes can be detected then the size of this vector would be 8×1 (p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3). The output from each grid cell is combined and with the configuration of current example, the shape of output becomes $3 \times 3 \times 8$. This can be observed in Figure 1.4. The value (a) amounts to 8 here. The output provided at the very end of this algorithm is the class of the object along with the coordinates of the bounding box.

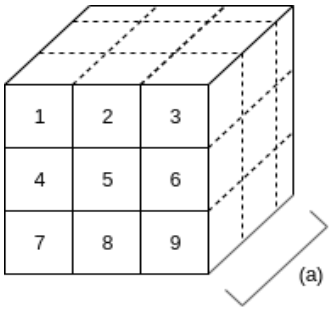


Figure 1.4: Size of output

1.4 Problem Statement

Over the years, object detection in images has been researched quite extensively and multiple methods have been developed. The research in the domain of object detection in videos has accelerated since the introduction of ImageNet challenge [29], [47]. It is noteworthy that videos are image sequences captured in a way that preserves the order of events as they unfold. Temporal information propagation when involved, helps to provide apriori estimates which aids in making coherent predictions which might not be the case when each frame is treated independently. In other words, videos offer the advantage of preserving temporal information when compared to images in real world challenging situations such as occlusions. The advantage of having temporal information is that it could be used to predict the next location of the object in case the detection is not possible [48]. Currently, many methods and techniques have been developed to detect objects in videos. These methods vary in their implementation and the technical approach adopted. For instance, some methods make use of image detectors in an iterative fashion whereas others make use of Recurrent Neural Networks (RNNs). A systematic compartmentalization of these approaches is needed. Also, it is of interest to investigate how different approaches harness the temporal information available in the videos to conduct object detection. This work aims:

- to provide a perspective of the approaches available to detect objects in video inputs including those which make use of temporal information.
- to discuss the benefits and drawbacks of the approaches.
- to implement selected methods/approaches and compare the performances (on custom data) based on certain parameters.

1.5 Challenges and Difficulties

Along with the benefits which videos offer, there are certain challenges and drawbacks which we have to overcome, some of the challenges are:

- As can be seen in Figure 1.5, videos suffer from issues such as motion blur, unfocused frames, partial occlusions and unnatural positions of objects [54]. Methods should have the ability to handle these issues successfully.
- Incorporating temporal information with the detection process can be difficult as some aspect of information flow is also involved. Additional processing might be needed for those methods that are able to tackle this problem successfully.
- Handling video inputs might be computationally intensive as compared to images. For instance splitting the video input into frames, generating the output and converting it back to video.
- Obtaining Ground truth for custom video data can prove tedious as every single frame constituting the video needs to be annotated separately. As length of the video increases, the number of frames increases and hence the time taken to annotate also increases.

Figure 1.5: Problems associated with processing videos [54]

1.6 Structure

This document has been organized in the form of six chapters. The chapter ***Introduction*** discusses the basic vision tasks with a focus on object detection. The motivation and challenges associated with a perception system are also discussed here. ***Theoretical Background*** deals with the concepts associated with the task of object detection. The discussions include prominent deep learning architecture (along with some auxiliary methods) and some domain specific terms which are referred to in the rest of chapters. In the chapter ***State of the Art and Categorization of approaches***, latest methodologies and approaches to tackle the problem of object detection in videos have been listed. It provides an overview of the solutions available to handle the problem of object detection. An attempt to categorize the discussed method has been made. Finally, a brief section in this chapter is dedicated to the discussions of dataset. ***Methodology of Experimentation*** describes the details of the experimental setup and the process adopted to conduct a comparative evaluation of different approaches of object detection in videos. These approaches are shortlisted from the discussions of previous chapter. The chapter of ***Results and Discussions*** analyses the results of the experiments conducted. It also discusses the anomalies and the possible reasons behind them. The final chapter ***Conclusion and Future Scope*** draws insights from the findings of the work. A short section in this chapter would also deal with the possibilities and future scope of this research work.

2

Theoretical Background

Object Detection forms backbone of many applications, hence a lot of research has been performed to make it more accurate and faster in the recent years. Various methods and approaches have been proposed to tackle the problem. To appreciate and understand the discussions in subsequent chapters it is imperative that the reader has a brief introduction to the Deep Learning architectures, namely the Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and the Long Short Term Memory (LSTM) Networks. Additionally, classical vision methods namely the Kalman filters and Optical Flow have also been discussed briefly to help understand the approaches discussed later. These concepts are used in the methods which are described subsequently.

2.1 Architectures

With developments in Deep Learning, many architectures have been proposed. There are a several architectures available for solving the problem of object detection. Some of the prominent network architectures for the task are Convolutional Neural Network (CNN) , Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) networks. Each has its own features which can be exploited to tackle the problem at hand. This section contains a brief overview of these architectures.

2.1.1 Convolutional Neural Network

This section discusses the basics of Convolutional Neural Network (CNN). As the name suggests, CNNs are the neural networks which make use of the *convolution* operation to learn features from the input data [20]. Generally, following are the types of layers associated with CNNs as can be seen in Figure 2.1:

- **Convolution Layer**
  This layer is useful for extracting features by making use of *convolutions*. The inputs to this

layer are image matrix and a filter matrix. The output of the convolution layer is called a feature map. Each filter is passed over the entire image and the resulting feature map for each filter is generated. The movement of the filter is determined by the *stride*. *Stride* is the

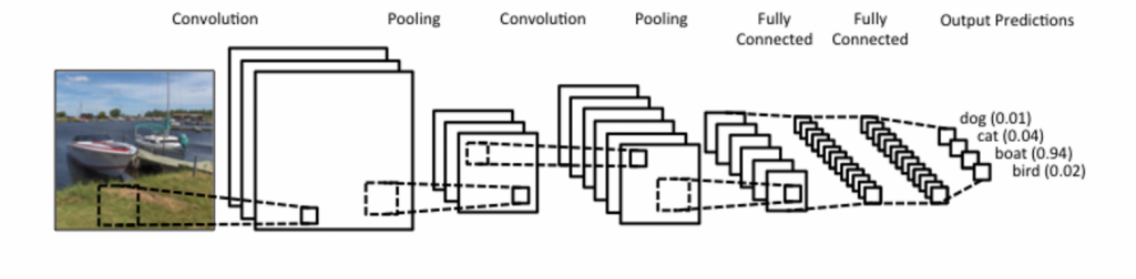


Figure 2.1: Convolutional Neural Network[1]

number of pixels by which the filter is shifted over the images. This is visualised in Figure 2.2. The filter is moved by 1 pixel hence the *stride* in this case equals 1. The size of the filter

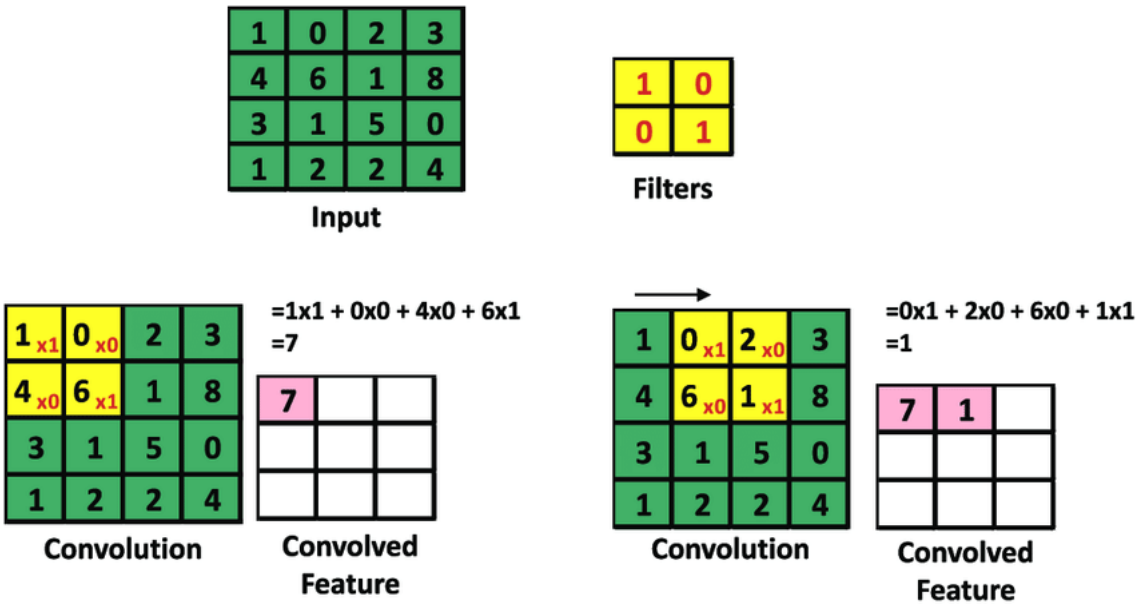


Figure 2.2: Convolution operation [39]

is 2×2. Size of the output of convolution is governed by the equation:

$$size\ of\ output = \frac{(size\ of\ input) - (filter\ size) + 2(padding)}{stride} + 1 \tag{2.1}$$

[1]Image adapted from: https://www.kdnuggets.com/2015/11/understanding-convolutional-neural-networks-nlp.html

Padding is an additional layer of pixels appended on edges of images in order to prevent loss of information while convolving. Additionally, it provides a possibility of preserving the dimensionality of feature map with respect to input. In the example demonstrated, the size of input is 4×4, filter size is 2×2, padding is 0 and the stride is 1. Substituting these values in Equation 2.1, the size of output obtained is 3×3 which can be verified from Figure 2.2.

- **Pooling Layer**
  The idea behind using a pooling layer is to learn the characteristic features in images. In order to do so, various mathematical operations can be used to learn the abstraction of the image. This would result in dimensionality reduction as well and help in training the network faster.
  - Max Pooling : Based on the *stride* and the *filter size*, the element which is maximum is selected from the feature map.
  - Average Pooling : As the name suggests, based on the *stride* and *filter size*, is the average of elements is selected from the feature maps.

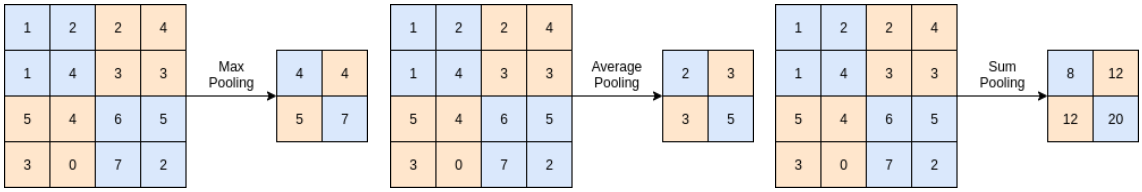


Figure 2.3: Types of Pooling[2]

  - Sum Pooling : This variant makes use of the sum of elements as opposed to maximum and average as discussed in the prior two cases.

- **Fully Connected Layer**
  In simpler terms, it down-samples the dimensionality while retaining the important information [6]. The different types of variants of Pooling can be seen in Figure 2.3. This example assumes filter size of 2×2 and a *stride* of 2: This layer outputs a vector having K dimensions where K is the number of classes that the network can predict. This contains the probabilities for each class.

These layers are the building blocks for object detectors. In the next part, one of the relevant frameworks built using these layers has been discussed.

Single Shot MultiBox Detector

One of the more popular frameworks to tackle the task of object detection in images is Single Shot MultiBox Detector [35]. Often referred to as SSD, it is a framework designed to handle the problem

[2]Image adapted from: http://cs231n.github.io/convolutional-networks/

of object detection using a single stage [35] unlike [19], [18] and [46] which have two stages for detecting objects where first stage acts as region proposal and the second serves the purpose of feature extraction and subsequent classification/detection. The model can be seen in Figure 2.4.

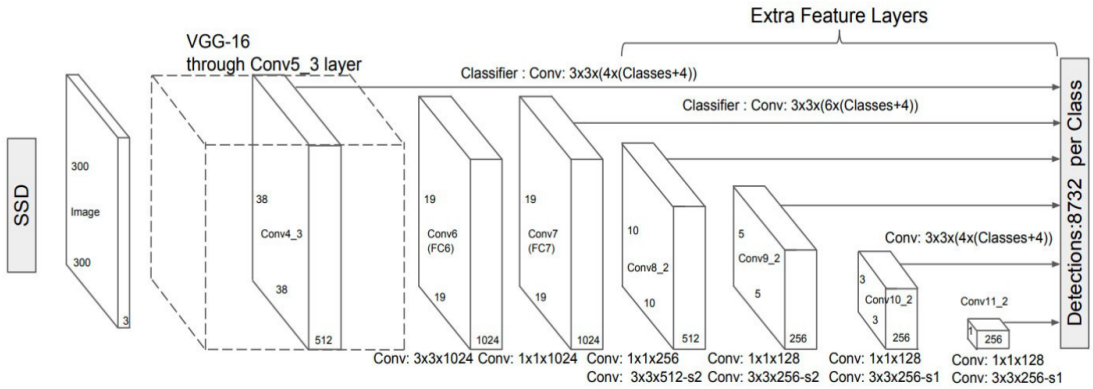


Figure 2.4: Single Shot Multibox Detector [35]

As the name suggests, the method takes only a single attempt to detect objects in the given input. The benefit in doing so is that the framework proposes the predicted boxes directly from the input without proposing the regions and hence saves up on time [27]. In Figure 2.4, it is seen that the base network (VGG-16) is used for extracting features and additional convolutional layers are used for detecting the objects [35].

VGG-16 is a CNN model [50] and the architecture can be seen in Figure 2.5. Additional convolutional layers are appended to this structure (to get the SSD architecture) and detection results are achieved. The size of these appended layers diminishes progressively allowing the predictions of the detection to work on multiple scales [35]. SSD computes the class score and locations using convolution filters. Once the feature maps are extracted, the filters are applied to obtain predictions [35].

[35] states that other base networks can be used in place of VGG-16. A variant of neural networks called as MobileNets [25] can be used. MobileNets are a category of efficient models for mobile applications making use of depth-wise separable convolutions to build light weight deep neural networks [25]. TensorFlow API[4] has a model repository in which an SSD framework using MobileNet as its classifier is provided.

2.1.2 Recurrent Neural Network

Convolutional Neural Networks (CNN) are better equipped to process data in the form of grids of values, such as an image [20]. Similarly, Recurrent Neural Networks (RNN) specialize in

[3]Image adapted from: https://neurohive.io/en/popular-networks/vgg16/

[4]https://github.com/tensorflow/models/tree/master/research/object_detection

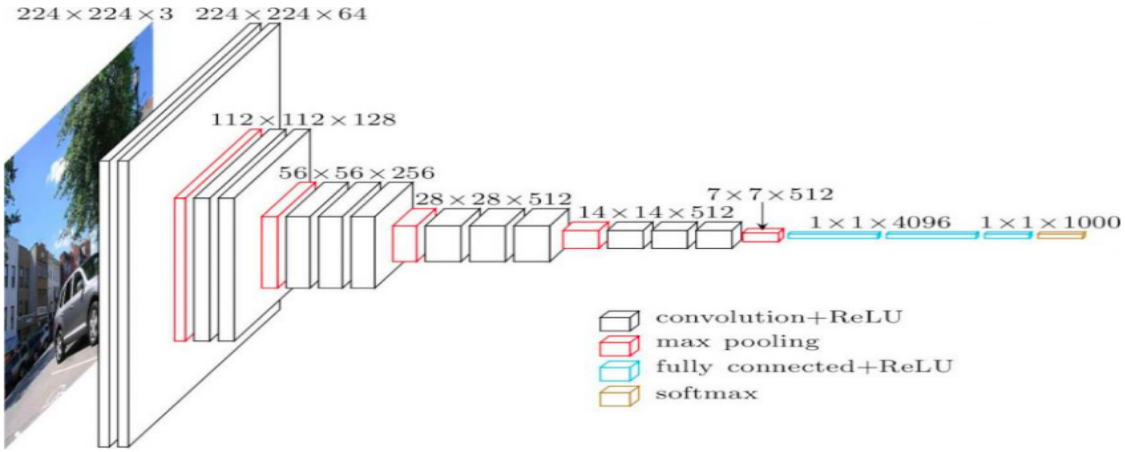


Figure 2.5: VGG-16 architecture [3]

processing temporal (or sequential) data. RNN achieve this by essentially employing *loops* which can be seen in Figure 2.6. Owing to this structure, RNNs have the ability to connect previous information to the present information which can be helpful for the object detection task in videos [7]. Information/features extracted from previous frame can be used to aid detection in current frame [7].

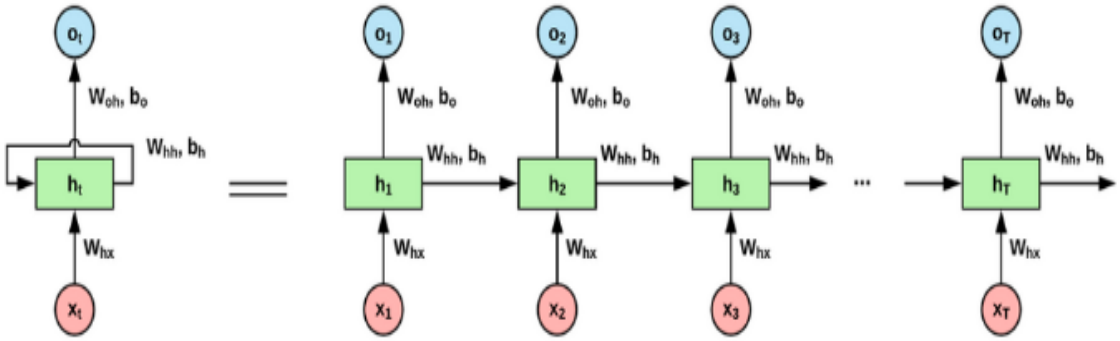


Figure 2.6: Recurrent Neural Network[5]

In Figure 2.6,
$\mathbf{o_t}$ = output state, $\mathbf{h_t}$= current time stamp, $\mathbf{h_{t-1}}$= previous time stamp, $\mathbf{x_t}$= current input, $\mathbf{h}$= single hidden vector, $\mathbf{W_{hh}}$= weight for previous hidden state, $\mathbf{W_{hx}}$= weight for current input state, $\mathbf{W_{oh}}$= weight for current cell state, $\mathbf{b_o}$= bias, $\mathbf{b_h}$= bias

Based on Figure 2.6, the mathematical understanding of single layer of RNN derived from [7] is described in brief. According to Equation 2.2, current state depends on past state and current

[5]Image adapted from: https://medium.com/@purnasaigudikandula/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9

input.

$$h_t = f(h_{t-1}, x_t) \tag{2.2}$$

The function generally used is *tanh* and hence the Equation 2.2 can be rewritten as Equation 2.3.

$$h_t = tanh(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{2.3}$$

The output can be expressed in terms of h_t as per the Equation 2.4.

$$o_t = W_{oh}h_t + b_o \tag{2.4}$$

However, it has been observed that as separation between the relevant information and the place where it is needed increases, RNN become less effective in establishing the learning [7]. This is referred to as the problem of *long term dependency*. This occurs due to vanishing (and exploding) gradient. In case of vanishing gradient, the learning stagnates as the gradient update during the backpropagation is not sufficient whereas in the case of exploding gradient, the gradient update is so large that the learning is not sensible [24]. The weights exhibit unstable variations and hence no useful learning can be achieved. This is resolved by using Long Short Term Memory (LSTM) networks [24] which are discussed next .

2.1.3 Long Short Term Memory

As described previously, RNNs suffer from the problem of learning *long term dependencies* [7]. To resolve this issue, Long Short Term Memory (LSTM) networks were introduced. The discussions and mathematical formulation described are taken from here [7]. The primary difference between RNN and LSTM can be seen in Figure 2.7 and Figure 2.8. The repeating modules in RNN have simple structure containing a *tanh* layer whereas the modules in LSTM have four layers interacting with each other. LSTMs have a *cell state* which is used for the information flow. LSTMs are equipped with the ability to add or remove some information to this cell state by making use of regulatory structures called as *gates*. Structurally, *gates* are made up of *sigmoid* layer and a pointwise multiplication operation [7].

The *forget gate* of an LSTM determines the amount of information to be *forgotten* [7]. It is governed by the Equation 2.5.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.5}$$

For value 0, complete disposal of information takes place and for value 1, complete retention of information takes place.

The *input gate* decides on what values from the current input needs to be updated. For this purpose, a *tanh* layer creates C_{it} as can be observed in Equation 2.6 and Equation 2.7. LSTMs update the old cell state C_{t-1} into the new cell state C_t. Equation 2.8 provides a mathematical

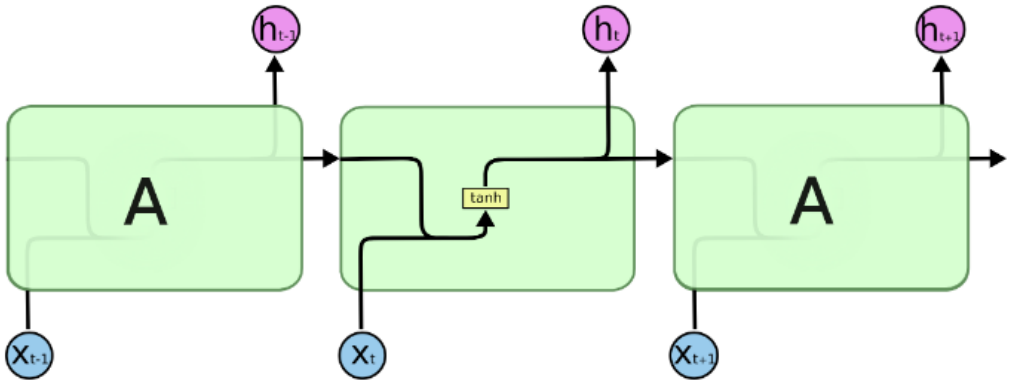


Figure 2.7: RNN internal structure [7]

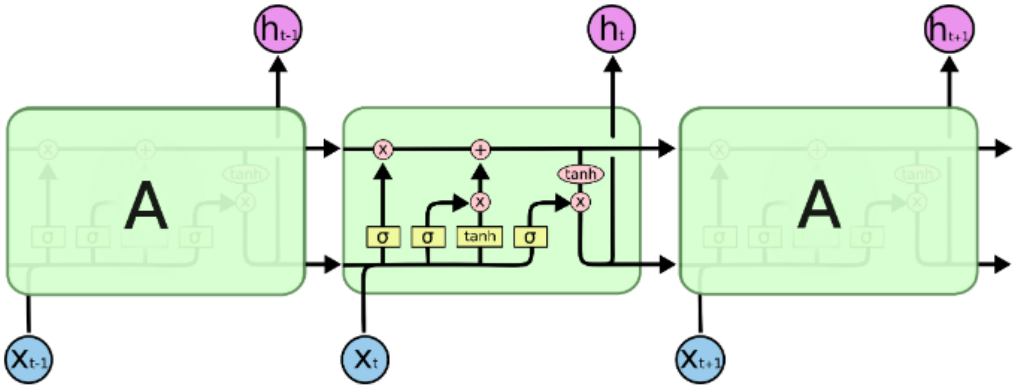


Figure 2.8: LSTM internal structure [7]

perspective of the concept.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{2.6}$$

$$C_{it} = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{2.7}$$

$$C_t = f_t * C_{t-1} + i_t * C_{it} \tag{2.8}$$

The *output gate* works as a filter for the cell state value. A *sigmoid* layer filters the cell state to determine the values for the output. A *tanh* is applied on the cell state calculated in the last step. Both of these values are multiplied to get filtered output according to our needs. Equation 2.9 and

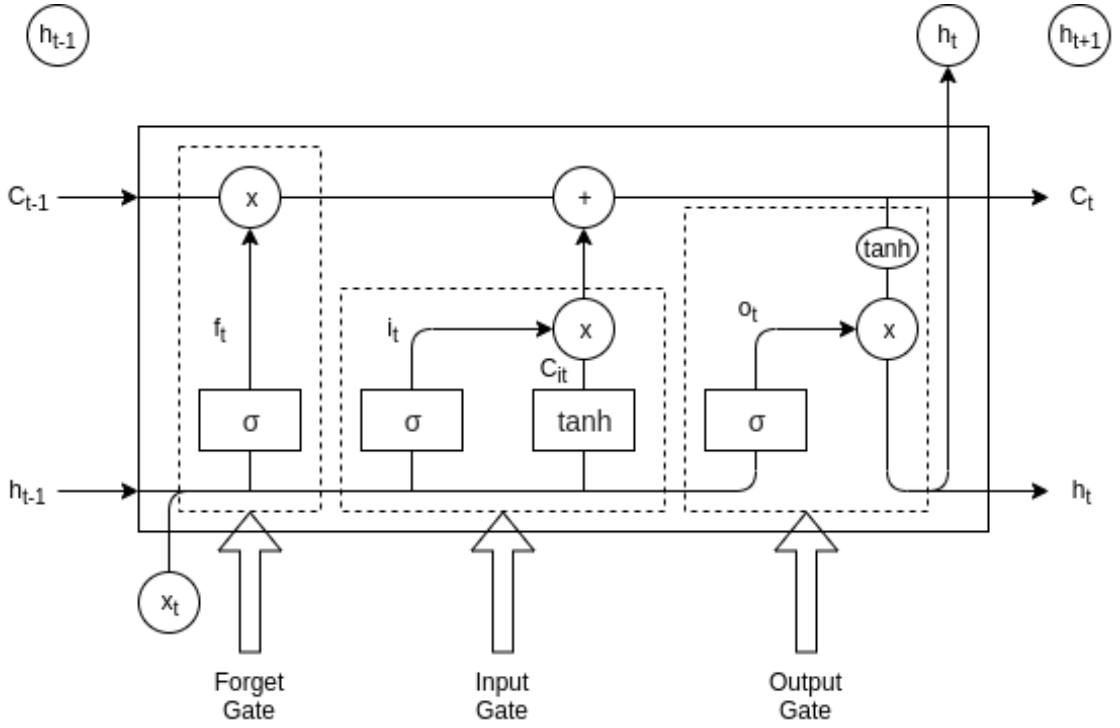


Figure 2.9: Gate structure of LSTM[6]

Equation 2.10 explains this concept mathematically.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{2.9}$$

$$h_t = o_t * tanh(C_t) \tag{2.10}$$

2.2 Auxiliary Methods

Several techniques serve as additional components for object detection which can be used in tandem with the deep learning. Auxiliary methods usually take lesser time to process than the detection algorithms. Using these approaches can help reduce the inference time as the videos do not exhibit drastic variations over consecutive frames. Two of such techniques discussed in this section are Kalman Filter and Optic Flow.

[6]Image adapted from: https://medium.com/@purnasaigudikandula/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9

2.2.1 Kalman Filter

The examples and discussions of this section have been derived from here [2], [3] and [4] . Kalman filter [26] is an algorithm that makes use of previous measurements of a system state, incorporates noise/errors in the system process and yields an estimate of unknown variables for the desired system state based on current measurements. Common applications of it are guidance, navigation, tracking of vehicles. Since it has such varied applications, the discussion only covers those aspects which are relevant to this work. The operation of filter comprises of two steps namely: predict and update. Figure 2.10 depicts a general overview of the algorithm.

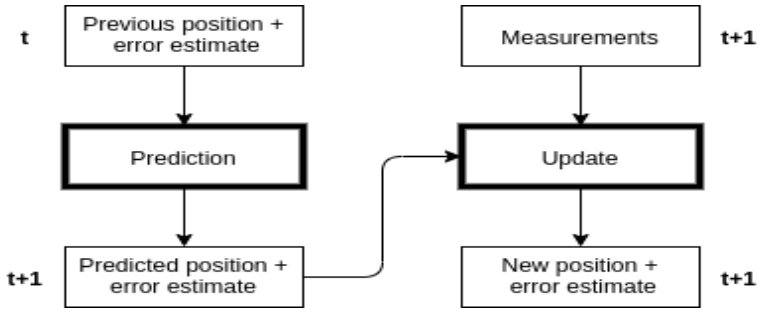


Figure 2.10: Kalman filter working[7]

As can be seen in the Figure 2.10, a prediction is made on the basis of previous values and error estimates (from time "t"). The prediction is based on a certain mathematical model (it can be constant velocity, constant acceleration or random motion). Along with the output of the prediction step, new measurements (from time "t+1") are combined to yield new values with an error estimate in the update step. This completes one cycle of the filter.

Mathematical analysis (taken from here [2]) of the filter can be summarised as:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - \hat{x}_{n,n-1}) \tag{2.11}$$

$$\hat{x}_{n+1,n} = \hat{x}_{n,n} + \Delta t \hat{\dot{x}}_{n,n} \tag{2.12}$$

$$\hat{\dot{x}}_{n+1,n} = \hat{\dot{x}}_{n,n} \tag{2.13}$$

$$K_n = \frac{p_{n,n-1}}{p_{n,n-1} + r_n} \tag{2.14}$$

$$p_{n,n} = (1 - K_n)p_{n,n-1} \tag{2.15}$$

$$p_{n+1,n} = p_{n,n} + q_n \tag{2.16}$$

where $\hat{\mathbf{x}}_{\mathbf{n,n}}$ = initial system state, $\mathbf{p}_{\mathbf{n,n}}$ = state uncertainty, $\mathbf{z}_{\mathbf{n}}$ = measured system state, $\mathbf{r}_{\mathbf{n}}$ = measurement uncertainty, $\mathbf{K}_{\mathbf{n}}$ = the Kalman gain, $\mathbf{q}_{\mathbf{n}}$ = the process noise and $\hat{\dot{\mathbf{x}}}_{\mathbf{n,n}}$ = the first order

[7]Image adapted from: http://www.argos-system.org/manual/3-location/32_principle.htm

derivative of x with respect to time. Equation 2.12 and 2.15 depends upon the system model used, here these equations are based on constant velocity system. One variant of Kalman filter works in the scenario when the measurements are spurios or completely absent. This type of filter is called Open Loop Kalman Filter (OLKF) [5]. In this variant, due to the absence of new measurements the update step is skipped and no Kalman gain is calculated. The predicted values are considered as the actual measurement in this variant. As it lacks the update step, this method is not as accurate as it would be when the method had an update step which is the case with the other variant. In the proposed approach, the measurement is assumed to be the prediction output from the deep neural network. However, the point of running adapted OLKF in tandem is to reduce the inference time, but running the detector at each step would defeat this purpose. Hence, this work tries to evaluate the adapted OLKF approach by using a suitable number of frames (determined empirically) when the detector should run and the detected bounding box values would be used in the update step.

2.2.2 Optical Flow

This section deals with a brief introduction to optical flow and its relevant mathematics. According to [11], Optical flow is the phenomenon of apparent motion of objects owing to the relative motion of a scene and an observer. The understanding and the discussions of this topic have been taken from OpenCV official documentation[8] and here[9]. Optical flow can be used to do motion estimation and object detection.

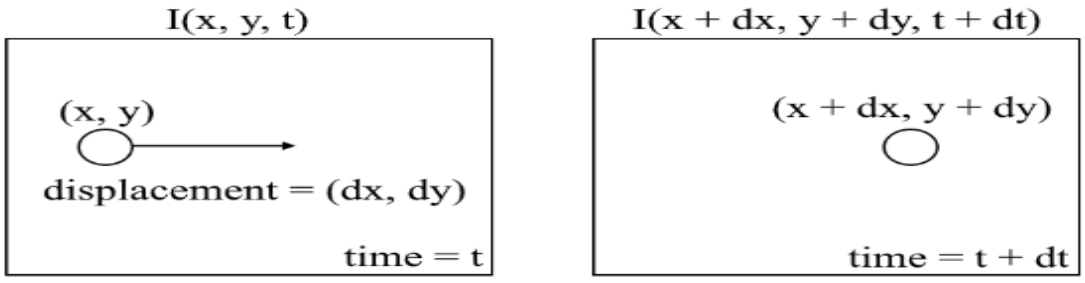


Figure 2.11: Understanding Optical flow[9]

A very basic understanding of optical flow can be visualised in Figure 2.11. Optical flow holds two assumptions namely:

- pixel intensity do not change over consecutive frames
- neighboring pixels exhibit identical motion

Based on the first assumption, the Equation 2.17 holds true.

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.17}$$

[8]https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html
[9]https://nanonets.com/blog/optical-flow/

Equation 2.17 can be rewritten as Equation 2.18 after applying Taylor series approximation.

$$f_x u + f_y v + f_t = 0 \tag{2.18}$$

where $f_x = \frac{dI}{dx}$, $f_y = \frac{dI}{dy}$, u = $\frac{dx}{dt}$ and $v = \frac{dy}{dt}$ f_x, f_y and f_t are image gradients and both u and v are velocity values. This can be solved using Lucas Kanade method [43]. Two types of optical flow estimations can be performed namely: Sparse and Dense. Sparse optical flow measure the velocity vectors only for a specific set of points and propagates that information to subsequent frames. On the other hand, in dense optical flow calculation, motion of all pixels in a frame is calculated.

Traditional methods to estimate optical flow included Lukas Kanade [43] method and Farneback's algorithm [15]. Recently some papers have attempted estimating optical flow using deep learning and [17] is one such work.

2.2.3 Insights

Both the techniques have certain positive aspects to offer. However, this work explores the working of Kalman filter in conjunction with deep learning architectures as:

- Optical flow estimation works with the assumption that the pixel intensity does not change over time. This might not be the case while testing on real life scenarios such as occlusions, variable lighting conditions etc.
- Kalman filter offers the alternative of working with missing/spurious measurements. Optical flow estimation might fail in such scenarios.

3

State of the Art and Categorization of Methods

3.1 Overview

The previous chapter introduced the basics of deep learning architectures and auxiliary methods which have been in use. This chapter discusses those approaches that focus on solving the task of object detection in videos. Research in video object detection has accelerated with the introduction of [47].

Videos are collection of images in cascade and the intuition to tackle the problem of object detection in videos is to repeatedly apply image detectors [35],[45],[18] on frames constituting the input video. The results obtained from this process would be considered as a reference to compare the performance of subsequent approach [54]. However, this approach does not incorporate the temporal information being offered in videos. As a step forward, combining object detection along with auxiliary methods is considered a natural progression. Lu et al. [37] briefly listed the possibility of using Kalman filter [28] or Particle filters [9].

Kang et al. [30] proposed a solution that employed *tubelets*[1] to make use of temporal information available in videos. The authors made use of still image detections in conjunction with *Motion Guided Propagation (MGP)* and *Tubelet Re-scoring*[2] to detect objects in videos. A method along the same lines was put forth by Han et al. [21]. The authors proposed to build sequences of bounding boxes from successive frames by using *Sequence Selection*[3].

Notably, some approaches applied aggregation and/or propagation techniques in order to harness temporal information. Zhu et al. [55] implemented image recognition network on selected frames (referred to as key frames) and propagated the derived features maps to remaining frames (referred

[1] This concept will be discussed in detail in subsequent sections
[2] 1
[3] 1

to as non key frames) using *flow field*[1]. Similar to this approach, [54] combined feature extraction network along with optical flow network to aggregate the feature maps and produce the detection results. Taking [55] and [54] as basis, [56] proposed three new techniques which improved the speed and the efficiency of feature computation. Hetang et al. [23] put forth an approach that drew inspiration from the human vision system. The authors utilised the fact that humans don't forget past frames and made use of an *impression mechanism* (combine features from key-frames in a weighted fashion and propagate the resulting *impression feature* along the length of video) in order to improve accuracy. It borrows the concept of flow guided feature propagation from the discussions of [17] and [55].

Feichtenhofer et al. [16] proposed a pipeline that jointly performs detection and tracking by detecting objects at a frame level and using track regression across frames.

Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) Networks have also been used in certain research works. Ning et al. [40] made use of YOLO [45] and LSTMs [24] in conjunction. The authors made use of YOLO to extract features and combined it with region information using LSTMs. Research by White et al. [34] investigated the possibilities of making use of memory modules (implemented using LSTMs) to improve accuracy of object detection in videos. The authors utilised convolution feature extractors interleaved with their lighter counterparts to identify the summary of the scene and the output of both types of extractors were fused using a memory module. Liu et al. [33] introduced an approach which applied SSD [35] and bottleneck LSTMs[4] to propagate feature maps across frames.

A particularly interesting combination of a classical feature extractor technique along with Kalman filter has been explored in [48]. The authors of this paper have applied feature extraction on video frames and processed its output using Kalman filters. The application of this method was in surgical assistance which included conditions like variable lighting, moving background and occlusion cases in the area to be operated. These situations can be encountered anywhere in real life. Furthermore, the authors claim that using Kalman filter enables this approach [48] to handle occlusion scenarios. This approach propagates temporal information without including complexities that other approaches present and hence can be considered for implementation (with modifications which are discussed in **Methodology of Experimentation**.

From the discussions of this section, it is decided that a per frame object detector and a coupled approach (incorporating detection and tracking) would be implemented in order to test their performances on real world videos. The details of implementation have been discussed in depth in **Methodology of Experimentation**).

3.2 Contemporary Methods

The previous section discussed the general trends that have been observed in the field of object detection in videos. The next section takes a closer look at some of the approaches that have been frequently discussed and referred to by researchers.

[4] variant of LSTMs that use depth-wise separable convolution to save computations

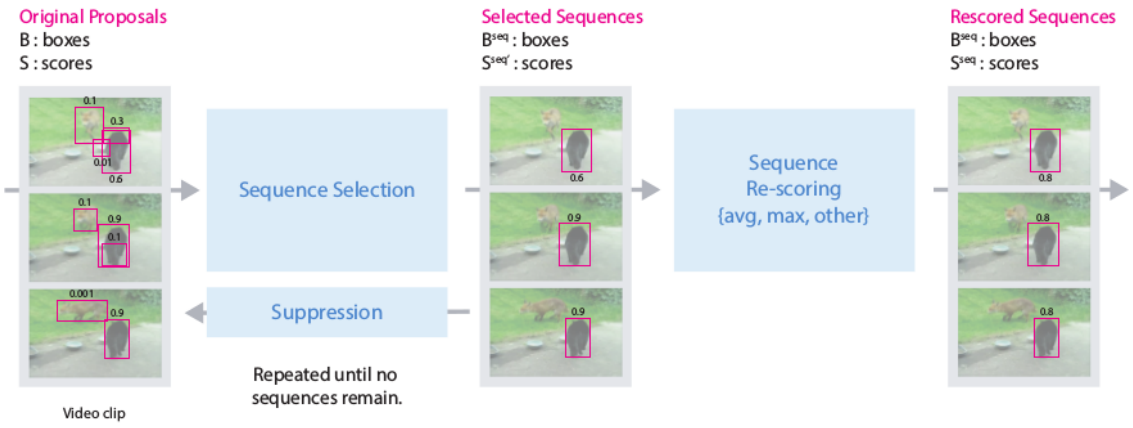


Figure 3.1: Seq-NMS [21]

3.2.1 Seq-NMS for Video Object Detection

The discussions in this section have been derived from Seq-NMS for Video Object Detection [21]. The authors suggest modifying the later stages of existing object detectors on images in order to make use of the temporal information. For this purpose, the object detector selected is [46]. The object proposals and the class scores obtained from the earlier stages of this detector are used by the suggested method to generate detections. The authors operate under the assumption that neighboring frames have similar objects and the corresponding bounding boxes exhibit temporal consistency, that is, the bounding boxes are similar in position and size. As can be seen in Figure 3.1, this approach consists of 3 steps namely:

- **Sequence Selection**
  The region proposal boxes for pair of neighboring frames are linked if the Intersection over Union (IoU) between them is greater than a threshold of 0.5. Such linkages are investigated throughout the video. The corresponding maximum score sequence (class score) is found for the entire video. Alternatively, authors attempt to obtain that sequence of region proposals that maximizes the sum of class scores under the restriction that all proposal boxes of neighboring frames are linked.

- **Sequence Re-scoring**
  Once the sequence is selected in the previous step, the score are improved by applying either *average* or *max* function on the score sequence.

- **Suppression**
  Suppression is applied within frames such that if a certain box has IoU greater than a threshold with the selected box, it is removed from consideration for the next iteration.

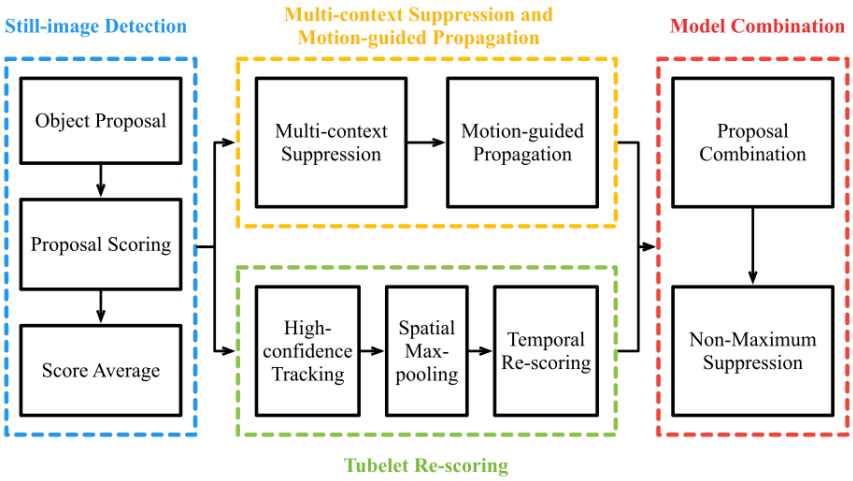


Figure 3.2: TCNN Framework [30]

3.2.2 TCNN:Tubelets with Convolutional Neural Network for object detection from videos

The discussions in this section have been derived from TCNN:Tubelets with Convolutional Neural Network for object detection from videos [30]. This work is based on the concept of *tubelets* which are essentially sequences of bounding boxes. The authors assume that if objects exist in a certain frame, they are likely to be present in the next frame in the vicinity of the original whereabouts. As can be seen in the Figure 3.2, this framework is composed of four components. The functioning of each component is explained in as follows:

- **Still-image Detection**
  Two types of frameworks are employed for still image detection, namely the DeepID-Net [41], which is an extension of R-CNN [19] and CRAFT [53], which is an extension of Faster R-CNN [46]. This component generates region proposals for objects for all frames in the input video. Furthermore, it assigns an initial detection score for each region proposal.

- **Multi-context Suppression and Motion-guided Propagation**
  Multi-context Suppression makes use of the fact that detections in each frames are highly correlated. The authors note that each training video usually contains selected number of classes which have correlation amongst them. This context information is used to suppress the *False Positives*. Furthermore, the authors rank all detection scores of the boxes in a decreasing manner for each video clip. The scores of low confidence classes (determined by a threshold) are suppressed by subtracting a certain value (both threshold and the subtracted values are obtained by analysing the validation set) whereas the scores of the high confidence classes are left untouched. Finally, the high scoring classes are selected based on the box

scores and the classes having lower scores are ignored.

Motion-guided Propagation uses the motion information in the form of *optical flow* to propagate detection results to neighboring frames. It is observed that using the motion information reduces *False Negatives*. *Optical flow* vector for each region proposal is calculated. Based on the mean flow vectors, the bounding box coordinates and the detection scores are propagated to the adjacent frames.

- **Tubelet Re-Scoring**
  The previous step generates short duration *tubelets* which helps reduce *False Negatives*. However, this only provides short term temporal consistency. In order to prolong the temporal consistency, authors suggest making use of tracking algorithms which help associate still image detections using tubelets for longer duration. Highly confident detection boxes called as *anchors* are selected as start point for tracking. Bidirectional tracking is conducted from the *anchor* using trackers described in [51] and a *tubelet* is constructed. Next process in this component is that of Spatial Max-Pooling, which essentially replaces tubelet box proposals with high confidence detections which are provided by still-image detector. The decision to replace depends on the IoU (between still-image detection and tubelet box) threshold. Higher the IoU, more confidence on the tubelet box. For instance, if IoU = 1, then full reliance on the tubelet box is assumed. Final process in this component is to classify the tubelets into two categories (positive and negative) and map the scores to difference ranges in order to increase the margin.

- **Model Combination**
  Results from DeepID-Net and CRAFT obtained initially are processed independently by applying tubelet re-scoring and motion guided propagation. The processed outputs of the two frameworks are then combined in this step by applying Non-Maximum Suppression to obtain the final results.

3.2.3 Deep Feature Flow for Video Recognition

The concepts and equations in this section have been taken from Deep Feature Flow for Video Recognition [55]. The idea presented by authors is to run the computationally intensive convolutional network intermittently and propagate the extracted features via flow field as the computation of features is time intensive when compared to propagation. As is with the previous two approaches, this method is inspired by the notion that consecutive frames in a video do not differ much. With reference to Figure 3.3, N_{feat} and N_{task} are two networks which are responsible for extracting features and performing detection respectively. The output shown in Figure 3.3 is that of semantic segmentation however the output for object detection can be obtained by modifying the final few layers. F is a flow estimation function which can be [17]. The authors of [55] claim that this approach is the first to train flow and video recognition tasks collectively. N_{feat} is selectively

applied on frames (referred to as key frames) and the remaining frames (referred to as non key frames) are used to calculate flow field ($M_{i\to k}$). The flow field is the output of [17] as can be seen in Equation 3.1.

$$M_{i\to k} = F(I_k, I_i) \tag{3.1}$$

$$f_i = W(f_k, M_{i\to k}, S_{i\to k}) \tag{3.2}$$

where $\mathbf{M_{i\to k}}$ = flow field, $\mathbf{F}$ = flow estimation algorithm, $\mathbf{I_i}$ = current frame , $\mathbf{I_k}$ = key frame index, $\mathbf{W}$ = feature propagation function, $\mathbf{f}$ = feature map, $\mathbf{S_{i\to k}}$ = scale function (estimates the scale variations between frames and adjusts the features accordingly) The propagation of temporal information can be seen at the juncture where the output of N_{feat} is being combined with the output of F. The authors make use of [17] for estimating the optical flow, [22] for feature extraction and [12] for detection tasks. Following describes the process overview:

Algorithm 1: Deep Feature Flow

```
accept input video frames
initialize key frame (k=0)
run feature extractor on key frame
run detection network on key frame
for input video frame do
    if is_key_frame then
        update the key frame (k=input video frame)
        run feature extractor
        run detection network
    else
        propagate features from key frames
        run detection network
    end
end
output detection result
```

3.2.4 Flow Guided Feature Aggregation for Video Object Detection

The discussions in this section have been taken from Flow Guided Feature Aggregation for Video Object Detection [54]. The authors suggest running feature extractor network on all frames.

To boost the feature extraction at a particular frame, the authors suggest using FlowNet [17] to incorporate the motion information between that particular frame and adjacent frames. Figure 3.4 summarizes the aforementioned discussions. As can be seen, features extracted at frame "t" are weaker as compared to "t-10" and "t+10" hence features from those frames are aggregated to improve the detection at frame "t". Feature maps from adjacent frames are combined with the current frame with the help of [17]. Accumulation of the feature maps from neighboring frame is done on the current frame on a weighted basis. One possible policy discussed by the authors is that of adaptive weight which prioritizes features from the frames near to current frame over those that

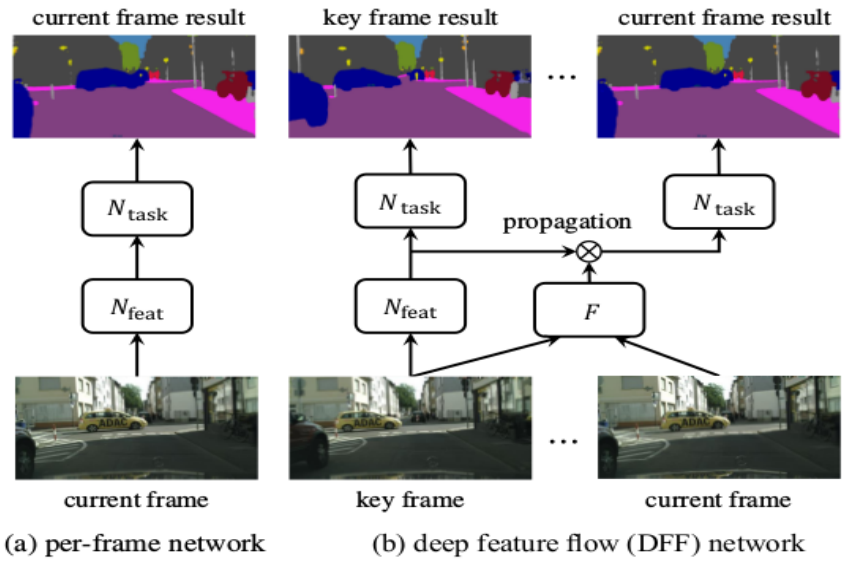


Figure 3.3: Deep Feature Flow [55]

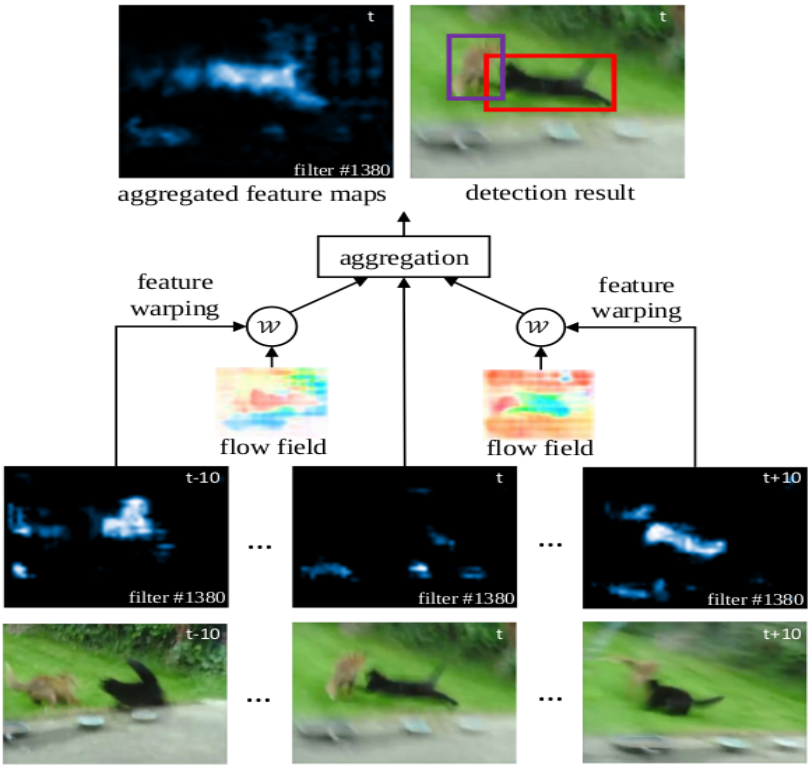


Figure 3.4: Flow Guided Feature Aggregation [54]

Algorithm 2: Flow guided feature aggregation

```
accept input video frames
for frames in feature buffer in range (1,K+1) do
    run feature extraction all frames falling inside the window
end
for input video frame do
    for frame# from max(1,i-K) to (i+K) do
        calculate flow between "frame#" and current frame
        compute embedding features/similarity measurement
        compute weights for aggregation based on policy
    end
    aggregate feature
    run detection on current video frame
    update the feature buffer
end
output detection result
```

are far and assigns the former higher weights. The authors have made use of [17] for calculating optical flow, [22] for feature extraction and [12] for generating detection results. The approach gives the flexibility to select the window of frames (also referred to as feature buffer, K) from which to aggregate features.

3.2.5 Towards High Performance Video Object Detection

The discussions in this section have been derived from Towards High Performance Video Object Detection [56]. Building on the discussions of [54] and [55], this work suggests new techniques which yields high performance for video object detection. The two works [54] and [55] discuss differing aspects of utilising temporal information but suffer from certain drawbacks. *Sparse feature propagation* discussed in [55] saves on complex computations by *propagating features* from key frames. Key frames are the selected group of frames on which computationally expensive feature computation network is implemented. This can be seen in Figure 3.5 (a). The drawback is that recognition accuracy is reduced owing to error during feature propagation. In [54], the authors discuss multi frame *dense feature aggregation*. As the name suggests, this improves feature quality by *aggregation* on every frame and this improves the detection accuracy. This is depicted in Figure 3.5 (b).

However, the drawback with *aggregation* on every frame is that complex computations are involved and it results in slowing down the approach. The authors in [56] propose three techniques which are essentially an extension of [54] and [55]. Following is a description of the methods:

- **Sparsely Recursive Feature Aggregation**
  This approach is a mixture of [54] and [55]. It calculates features and aggregates them at sparse key frames. This can be seen in Figure 3.5 (c1). As can be seen, the history of all key frames is propagated to and aggregated on next key frame.

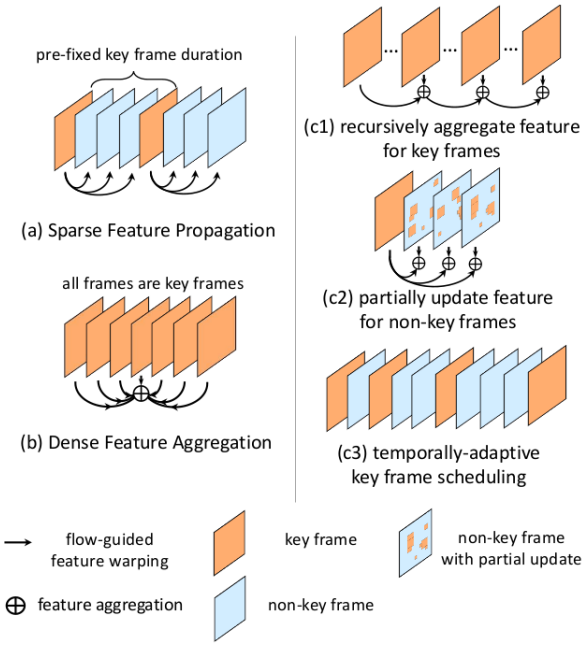


Figure 3.5: Flow-based methods [[54] and [55]] and their proposed extensions [56]

- **Spatially-adaptive Partial Feature Updating**
  A disadvantage of [55] is that the quality of propagated feature might be affected and might compromise the detection results. This approach addresses the drawback of [55] by computing a *feature temporal consistency* metric which quantifies if propagated feature is a good approximation of actual feature. To do so, a flow network is used to predict the motion field. If the *feature temporal consistency* is less than a threshold, the propagated feature is updated with the actual feature. The partial update step can be seen in Figure 3.5 (c2).

- **Temporally-adaptive Key Frame Scheduling**
  This approach makes use of the *feature temporal consistency* to decide the key frame scheduling. For a given frame, if the area to be recomputed (based on *feature temporal consistency*) is larger than a threshold, the frame is marked as a key frame. This is depicted in Figure 3.5 (c3).

Similar to [55] and [54], authors of [56] have made use of [17] to estimate optical flow, [22] for feature extraction and [12] for detection.

3.3 Analysis of the Contemporary Methods

The previous section described the contemporary methods in detail. This section discusses the similarities and differences between the approaches.

From the discussions, it can be inferred that the [21] and [30] have several similarities. Both approaches make use of *sequences* and *tubelets* respectively which are essentially series of bounding boxes to propagate temporal information. Both incorporate temporal information as a modification to later stages of existing object detectors. Each individual frame of the input is processed by detector networks. A critical observation is that temporal information is not learned in both these approaches rather obtained by stringing detection box sequences. Many research works for instance [49], [56], [10] refer to these approaches as *Box level* methods as they deal at the level of bounding boxes.

Advantages of Box Level Methods

- No requirement of additional neural networks (sub networks) which estimate optical flow or aid in feature propagation.

Disadvantages of Box Level Methods

- [31] claims these methods tend to be unsuitable for real time applications owing to huge computational requirements.
- These methods work on individual frame level and hence might fail in the case of prolonged occlusion. *Sequences* and *tubelets* would suffer discontinuity at such instances.
- These are multistage pipelines and are prone to failures as results of later stages are dependent on performances of earlier stages [54].

Similarly, [55], [54] and [56] exhibit commonalities though there are substantial differences as well. As discussed earlier, [56] extends the concepts elaborated in [55] and [54]. All three approaches employ motion estimation model for feature propagation and incorporate end to end learning of all sub components. [49], [56], [10] refer to these approaches as *Feature level* methods as they utilise features to handle temporal information. All three methods utilise the same sub-components as they make use of [17], [12] and [22] in their processing. [55] does not extract features on all frames whereas [54] does. On one hand, doing so allows [55] to ease the computational loads as compared to [54] which has to extract features on all frames. On the other hand, [55] detection results might be compromised owing to errors in feature propagation whereas [54] benefits from the rigorous feature extraction and aggregation [56].

Advantages of Feature Level Methods

- Since the flow propagation techniques are learnt using neural networks, these approaches are better equipped to handle occlusion as the approach does not encounter discontinuity as was in the case of *Box level* approach

- Affords a range of possibilities/trade-off in that [55] allows for lesser computation by selecting key frames, [54] enables the aggregation of features at every frame in order to maintain the sanity of detection results. [56] takes this a step further by allowing to work with an adaptive key frame scheduling policy, selective feature updating and recursive feature propagation and aggregation (see Figure 3.5).

Disadvantages of Feature Level Methods

- Additional requirement of having the sub-networks be trained end to end rather than being trained separately. The end to end training implies that the flow information should also be learned by the flow network at the time of training.

3.4 Categorisation of Video Object Detection Methods

This section attempts a systematic categorization of the methods discussed. It should be noted that segregating researches into the defined categories might not be entirely accurate as ideally every paper offers an unique perspective on how to tackle the problem of object detection in video. After analysing the contemporary research methods, the categories to be considered for this classification are:

- **Temporal Information Usage:**
  This category segregates the approaches on the basis of how the temporal information is incorporated. *Box Level* or *Post Processing* methods are those which propagate the temporal information as sequences of boxes and rely on stitching bounding boxes. The other category of approaches that rely on feature maps are listed under *Feature Level*.

- **Framewise feature extraction:**
  This category is based on whether the method employs feature extractor network on each frame of the input video or not.

- **Approach(Paradigm):**
  Those methods which employ auxiliary methods for instance classical optical flow calculation or trackers in the form of kalman filters etc are listed under *Deep Learning + Classical*. Whereas those methods in which auxiliary methods are also trained using deep learning are listed under *End to End Deep Learning* methods. A third category titled *End to End Classical* method has been included which includes those researches which have tackled this problem using non deep learning methods.

Method	Category						
	Temporal Information Usage		Framewise Feature Extraction		Approach		
	Box Level (Post Processing)	Feature Level (During Training)	Yes	No	Deep Learning + Classical	End to End Deep Learning	End to End Classical
Sequence NMS for video object detection [21]	✓		✓		✓		
TCNN : Tubelet with CNN for object detection in videos[30]	✓		✓		✓		
Deep Feature Flow for Video Recognition [55]		✓		✓		✓	
Flow Guided Feature Aggregation for video object detection [54]		✓	✓			✓	
Towards High Performance video object detection [56]		✓		✓		✓	
Impression Network for Video Object Detection [23]		✓		✓		✓	
Video Object Detection with an AlignedSpatial-Temporal Memory [52]		✓	✓			✓	
Mobile Video Object Detection with Temporally-Aware Feature Maps [33]		✓	✓			✓	
Recurrent Residual Module for Fast Inference in Videos [42]		✓	✓			✓	
Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking [40]		✓	✓			✓	
A Kalman-Filter-Based Common Algorithm Approach for Object Detection [48]	✓			✓			✓

Table 3.1: Categorization of Video Object Detection methods

Table 3.1 offers interesting insights into the trends of object detection in videos. [21] and [30] are the only methods which fall under the *Box level* category as they process *sequences* and *tubelets*. Since these two methods make use of classical approaches for conveying temporal information , these works are listed in *Deep Learning + Classical*. These also implement feature extractor networks on all individual frames. [30] uses classical techniques like optical flow to link temporal information. [55], [56] and [23] methods are categorized identically since they apply feature extractor network on selected frames (referred to as key frames) and propagate these extracted features to the remaining frames (non key frames). [23] emulates the human vision system by retaining an impression of features derived on past key frames. This process aids in boosting detection of those frames where weaker features were extracted. [54], [52], [33], [42] and [40] exhibit identical categorization traits. [33] incorporates LSTMs to utilise temporal information. Convolution LSTM layers are inserted in image detection frameworks in order to convey frame specific information. [40] makes use of LSTMs [24] and YOLO [45] in conjunction. Visual features are extracted on individual frames and therefore this method has been correspondingly categorized. [52] utilises a memory module that aggregates information along time axis. The memory module is essentially an RNN architecture. As discussed earlier in **State of the Art**, [54], makes use of [17] in order to learn optical flow therefore, this has been listed under *End to End Deep Learning*.

An important observation while discussing this table is that the term *Box Level* can be interchangeably used with *Post Processing*. This is due to the reason that the temporal information is used *after* processing of main network as is in the case of [21] and [30]. Such methods link bounding boxes rather than dealing with feature maps and hence are termed as *Box Level* methods. Table 3.1 serves as a look up reference in situations when the requirements/constraints that the problem at hand sets are available.

3.5 Discussion of Datasets

Datasets are a vital component in any deep learning pipeline. It is of utmost importance that the data available for the task at hand should be systematic, clean and properly labeled. This section provides a brief overview of the prevailing datasets and would discuss their different aspects. Owing to the drastic developments in the field of object detection, a number of datasets have been developed. For the task of video object detection, it is vital to study the datasets comprising solely of images. This is due to the reason that some approaches for video object detection [30], [34], [37] make use of image detectors trained on image datasets as the starting point. Following are the list of datasets which will be discussed briefly in this section:

- PASCAL VOC [14]
- ImageNet [47]
- MS COCO [32]
- YouTube Bounding Boxes [44]
- CORe50 [36]

3.5.1 PASCAL VOC

PASCAL VOC stands for **P**attern **A**nlysis, **S**tatistical Modelling and **C**omputational **L**earning **V**isual **O**bject **C**lasses. The distribution of objects by class in this dataset can be seen in Figure 3.6. The designers of the dataset had the aim to include various object sizes, illumination conditions,

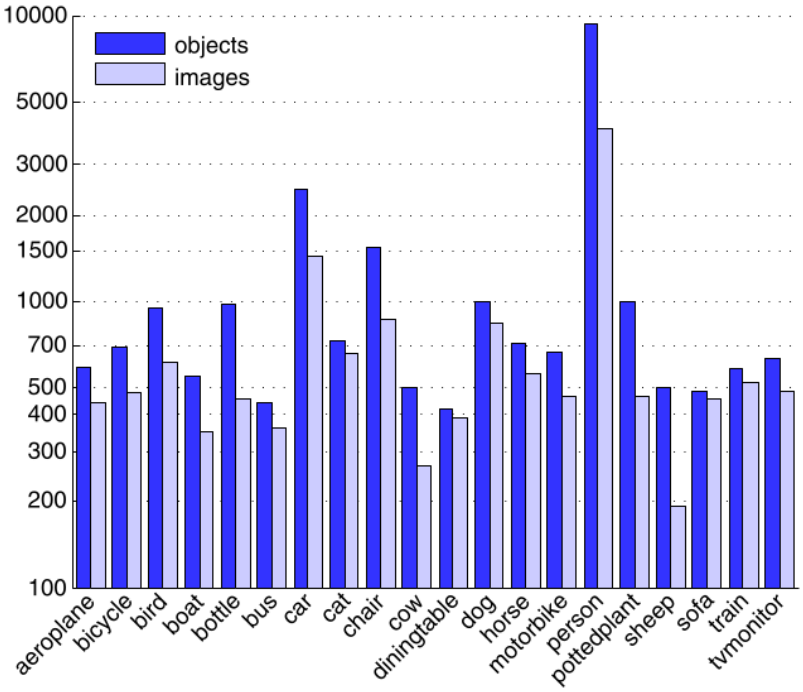


Figure 3.6: Class-wise distribution of objects in VOC dataset [14]

occlusion and feature space [14]. The dataset contains roughly 9,960 annotated images and 24,640 annotated object instances belonging to 20 classes [14].

As can be seen in Figure 3.6, this configuration yields an average of 1,232 objects per class. The metrics for evaluation discussed in [14] are *Average Precision* and *Bounding Box Evaluation*. The *Average Precision* is calculated by making use of mean precision of eleven equidistant recall levels and the *Bounding Box Evaluation* is calculated using *Intersection over Union* of predicted bounding box and the ground truth bounding box [14].

3.5.2 ILSVRC

ImageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge (ILSVRC) is a competition which has been organized for classification and detection tasks [47]. Two variety of annotations have been discussed in [47] namely image level annotation and object level annotation. The former essentially yields output like a classification task and the latter provide the output like an object detection

task [47]. The dataset for classification task contains a total of 1000 object classes and has roughly 1,350,000 images split into train (1,200,000 images), validation (50,000 images) and test (100,000 images) sets [47]. For the task of object detection, [47] splits the 200 classes in 450,000 training images, 20,000 validation images and 40,000 test images (total 510,000 images). ImageNet [13] forms the basis for [47]. In this competition a separate task was introduced for object detection in videos called Imagenet VID [47], [21]. Based on the numbers provided by [21], the training set contained approximately 3800 clips consisting of 1,122,397 frames and the validation set contained roughly 555 clips with 176,100 frames and 30 classes of objects.

3.5.3 MS COCO

MS COCO stands for **M**icrosoft **C**ommon **O**bject in **CO**ntext. The creation of the dataset [32] had the objective to answer the question of scene understanding. To resolve this, [32] addressed the underlying issues of detecting objects in their non iconic views (unusual perspectives), reasoning of context between objects and accurate localization of objects in 2 dimensions. A total of 2,500,000 labeled instances were annotated in 328,000 images in the dataset. The objects were distributed over 91 categories. As can be seen in Figure 3.7, 82 of the 91 total classes have over 5,000 object

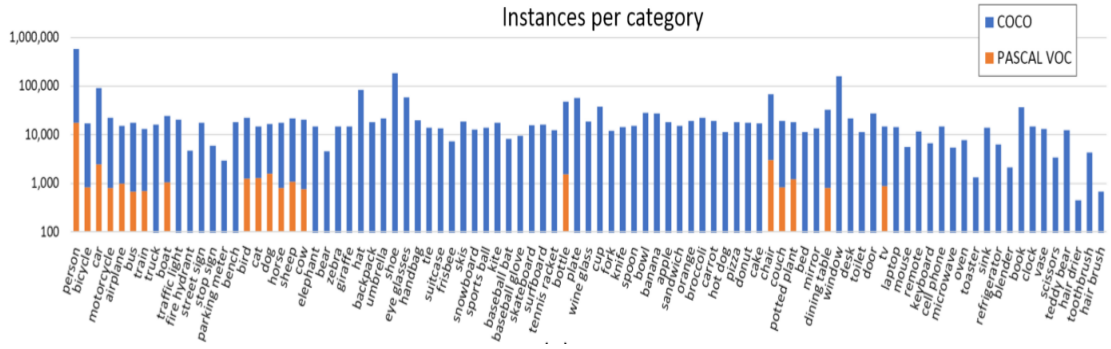


Figure 3.7: Class-wise distribution of objects in MS COCO dataset [32]

instances. A roughly homogeneous distribution of instances per category can be seen across all the categories in the dataset.

One interesting observation is that COCO contains approximately 3.5 categories and 7.7 object instances per image. In contrast, both ImageNet and PASCAL VOC have less than 2 categories per image and 3 instances per image [32]. ILSVRC paper states that the MS COCO is likely to evolve into an important large-scale benchmark [47].

3.5.4 YouTube Bounding Boxes

YouTube Bounding Boxes is a dataset containing video URLs. It contains annotated bounding boxes [44] for objects. The objects are included in this dataset are a subset of those included in the

COCO [32] dataset. In total, 23 classes of objects have been included. Approximately 5,600,000 bounding boxes have been drawn in 380,000 video segments [44]. The dataset [44] employs a strategy of subsequently increasing precision of human annotations which finally results in an annotation accuracy of 95% for every class.

3.5.5 CORe50

CORe50 stands for **C**ontinuous **O**bject **Re**cognition of 50 domestic objects belonging to 10 classes [36]. The videos in this dataset were recorded by gradually moving objects in front of the camera. This added the temporal coherence to the data. The concept of temporal coherence was introduced in [36] to address the lack of availability of multiple perspectives of same object in a variety of background. The data is collected in 11 sessions which are recorded under varying lighting and background conditions [36]. The dataset is available for download in the form of images (extracted from the recorded videos). A total of nearly 164,866 images are contained in the dataset [36].

3.5.6 Summary of datasets

The previous section provided a brief overview of datasets and described the statistics. This section compares these statistics and draws some insights. Table 3.2 provides a summary of the statistics discussed earlier.

Dataset	**# of Categories**	**Annotated Instances**	**# of Images/Videos**
PASCAL VOC	20	24,640	9,960
ILSVRC (Object Detection)	200	534,300	510,000
MS COCO	91	2,500,000	328,000
YouTube BB	23	5,600,000	380,000
CORe50	10	164,866	164,866

Table 3.2: Comparison of datasets

Figures 3.8, 3.9 and 3.10 provide us insights into the statistics presented in Table 3.2. From Figure 3.8, it can be inferred that MS COCO has the highest number of object categories per image amongst the analysed datasets[5]. Also, the MS COCO includes more images of objects under scenarios of occlusion and clutter (these are considered difficult from the perspective of object detection) [32]. From Figure 3.9, it can be seen that MS COCO has the appropriate balance between the number of categories included and the instances per category.

Though, YouTube BB has a much higher instances per category count, it has considerably lesser number of object categories included. This might result in strong learning for those object classes. Also, the possibility that a video might be removed[6] from the dataset persists rendering the dataset

[5]The number of object categories per image for YouTube BB dataset is not specified.
[6]https://research.google.com/youtube-bb/download.html

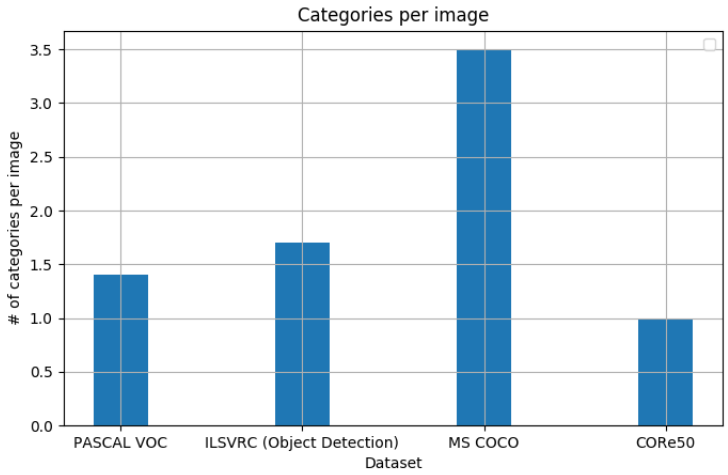


Figure 3.8: Number of categories per image

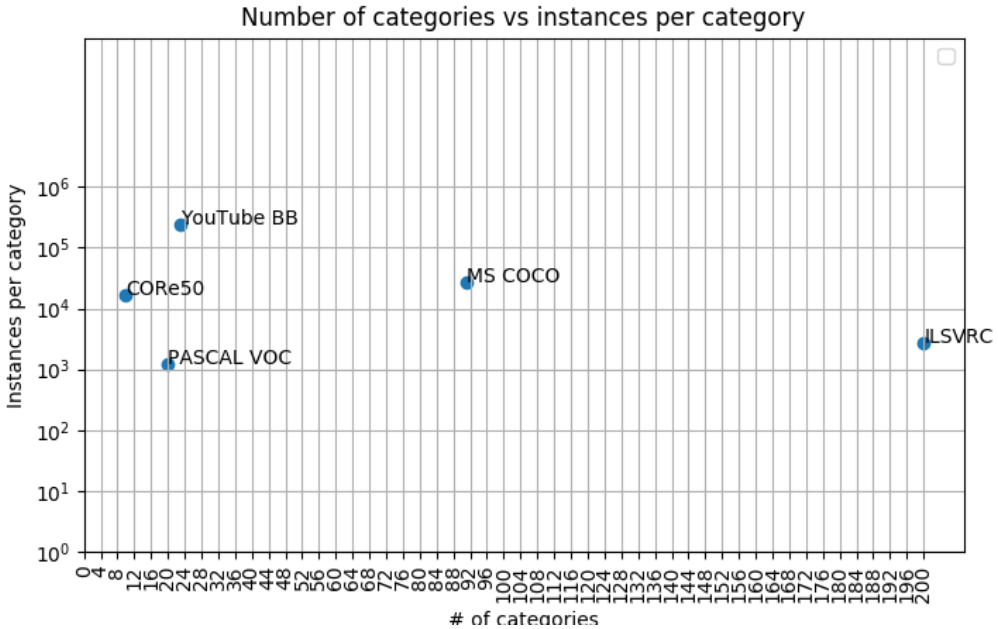


Figure 3.9: Number of categories vs instances per category

vulnerable to instability. The issue of lesser number of object categories persists with PASCAL VOC and CORe50 which have even lesser number of object categories. On the other hand, though ILSVRC has larger number of categories, the number of instance per category is significantly lesser that MS COCO.

From Figure 3.10, it is clear that of the image datasets, MS COCO has the highest count

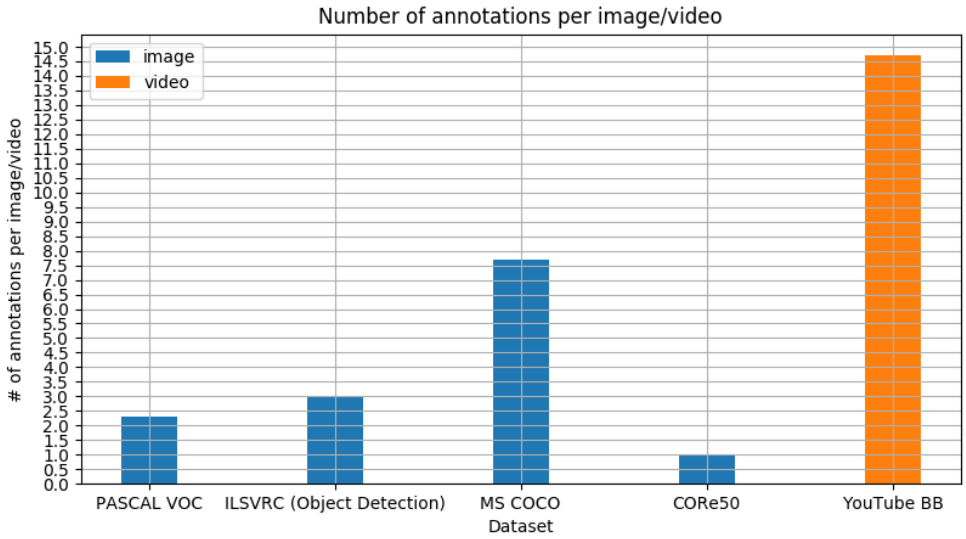


Figure 3.10: Number of annotations per image/video

of classes per image when compared to PASCAL VOC, ILSVRC and CORe50. However, one interesting thing to note here is that YouTube BB has nearly double the number of annotations per image/video of MS COCO but the annotations for the YouTube dataset are performed for video segments rather than images. In [44], it is stated that 5 frames are considered from each video segment. Incorporating this fact into the calculation provides a valid comparison platform as this yields a value of 2.94 instances per frame which is lesser than the count of MS COCO. The primary reason that leads to use the MS COCO dataset is that pre trained weight files for networks trained on this dataset are readily available on TensorFlow[7] and since the primary objective of this research work is to evaluate the approaches rather than dealing with the complications of training a network from scratch. Summarising from the discussions conducted above, MS COCO seems to be the optimum choice amongst the datasets discussed.

[7]https://github.com/tensorflow/models/tree/master/research/object_detection

4

Methodology of Experimentation

This chapter discusses the details of the methodology that has been used to conduct a comparative evaluation between different approaches for object detection in videos. The approaches along with the pipeline of the implementation is also discussed. A description of the hardware and software specifications is also provided. A methodology adapted for recording test data has been discussed.

From the discussions of Chapter 3, two approaches were shortlisted to conduct the comparison. A network pre-trained on the Microsoft Common Objects for Context (MS COCO) [32] has been used for the purpose of this evaluation. MS COCO is the dataset selected for this purpose for the reasons discussed in the closing stages of Chapter 3.

The base network used for object detection is **Single Shot MultiBox Detector** [35],[25] which has been pre-trained (on MS COCO [32]).

As discussed in earlier sections, object detection in images has been extensively researched. However, object detection in videos has gained attention with the introduction of [47]. Based on discussions in [31], [16], [54], this section aims to analyse the approaches to conduct object detection in video. The grounds for comparison are Intersection over Union and Inference Time which would be discussed in this chapter in detail.

4.1 Description of Approaches and Pipeline

This section contains a description of the approaches adopted for the purpose of comparison.

4.1.1 Framewise approach

Videos can be visualized as seen in Figure 4.1.

Using this fact, the individual images constituting the videos can be processed as collection of images in cascade to tackle the problem of detection which can be seen in Figure 4.2. An individual image is passed to the object detection neural network which provides an output in the form of a class of the object along with the bounding box coordinates. The object detection would

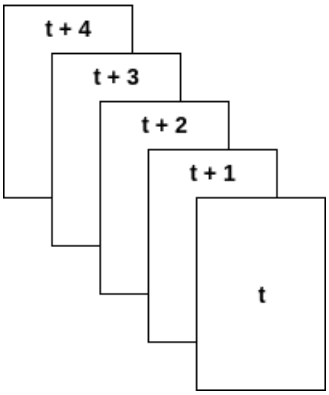


Figure 4.1: Videos can be considered as a collection of frames in cascade

be accomplished by making use of **SSD MobileNet[35]** (made available by TensorFlow API[1]). Scripts were written in order to facilitate the functioning of the pipeline. Customized annotation were done on the individual frames using a script written in OpenCV to handle the video frames. Also, a script written in Python provided the value of Intersection over Union of a pair of bounding box (Ground Truth annotation and the output of the approach) given the input coordinates for both the boxes. These scripts have been discussed in the **Appendix C**.

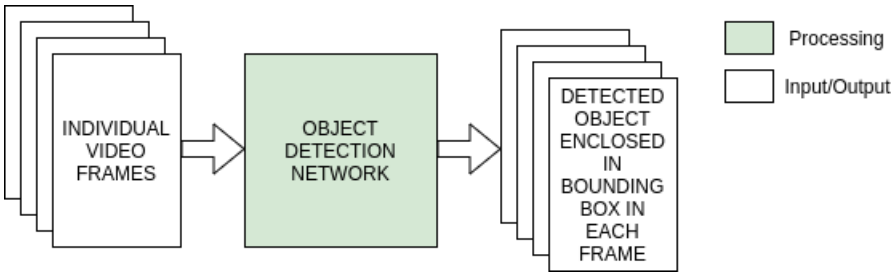


Figure 4.2: *Framewise* Detection Approach

An overview of the pipeline can be seen in Figure 4.3. The steps (d), (e), (f) and (g) are enclosed in a loop structure. These are repeatedly executed unlike (a), (b) and (c). Following is the description of this pipeline:

- Videos were recorded under varying conditions and scenarios. These conditions have been discussed in the Section 4.3 **(Test Data Specifications)**.
- Manual annotation of the videos frames (50 frames per video) using a custom script.
- Modification of the TensorFlow API to accept input in the form of videos and process it in the form of individual images. A functionality is added such that the API accepts the input

[1]https://github.com/tensorflow/models/tree/master/research/object_detection

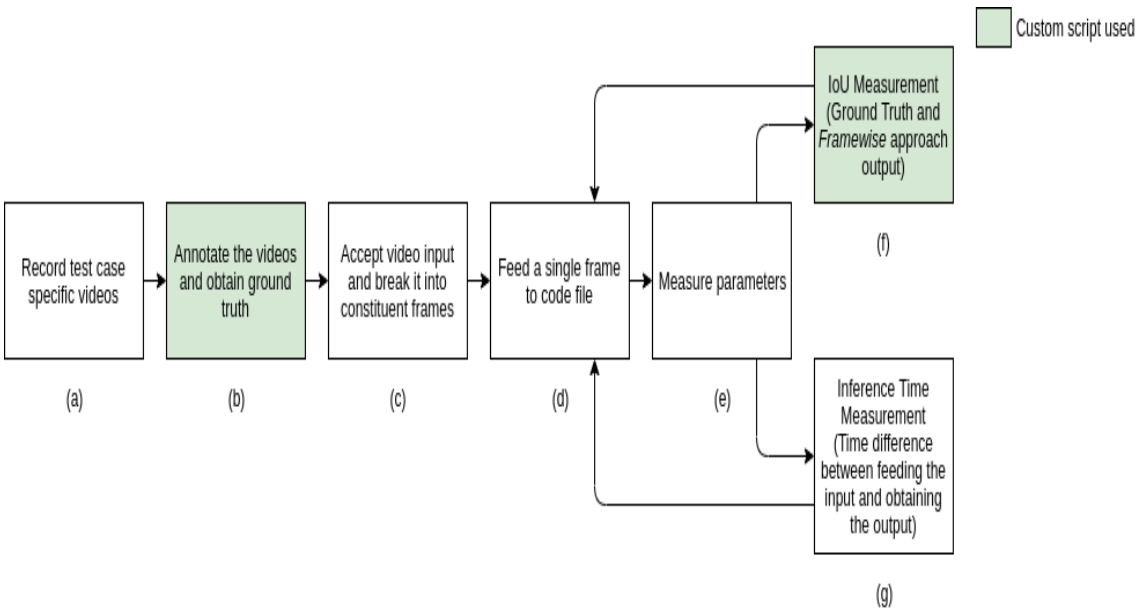


Figure 4.3: Pipeline adopted for *Framewise* approach

as video, disintegrates it into individual frames and then processes the frames individually. A diagrammatic description of this is provided in Figure 4.4.

- Calculating the parameters. A detailed description of the parameters listed below has been provided in Section 4.2 **(Selection of comparison parameters)**.
  - For calculating the Intersection over Union, record the output values of step (d) (these would be the bounding box coordinates) and store them in a file. Making use of the coordinates obtained from the manual annotations (obtained from step (b)) and the output coordinates (obtained from step (d)), calculate the Intersection over Union. A script has been developed for the calculation of Intersection over Union, provided two sets of bounding boxes coordinates. A custom script was developed as the format of the bounding box coordinates extracted were in a different format than the ones generated by TensorFlow API.
  - For calculating the Inference Time, a functionality has been added to the API which measures the amount of time taken to execute the code file. It is measured from the processing of the first frame till the time the last frame has been processed. In addition to recording the end to end Inference Time, an operation specific time has also been measured. The latter can be used to analyse how much time the code takes to process a single frame (in other words, time taken to process a single *Detection* operation). The graphs for the operation specific time have been plotted in the **Appendix B**.
- Once the parameters (IoU and Inference Time(operation specific)) are measured for one frame,

the customization made to the API in step (c) enables the code file to accept the next input frame and follow the procedure again.

Figure 4.4 provides an insight into the blocks (e), (f) and (g) of the Figure 4.3 and provides a diagrammatic representation of flow of the code file.

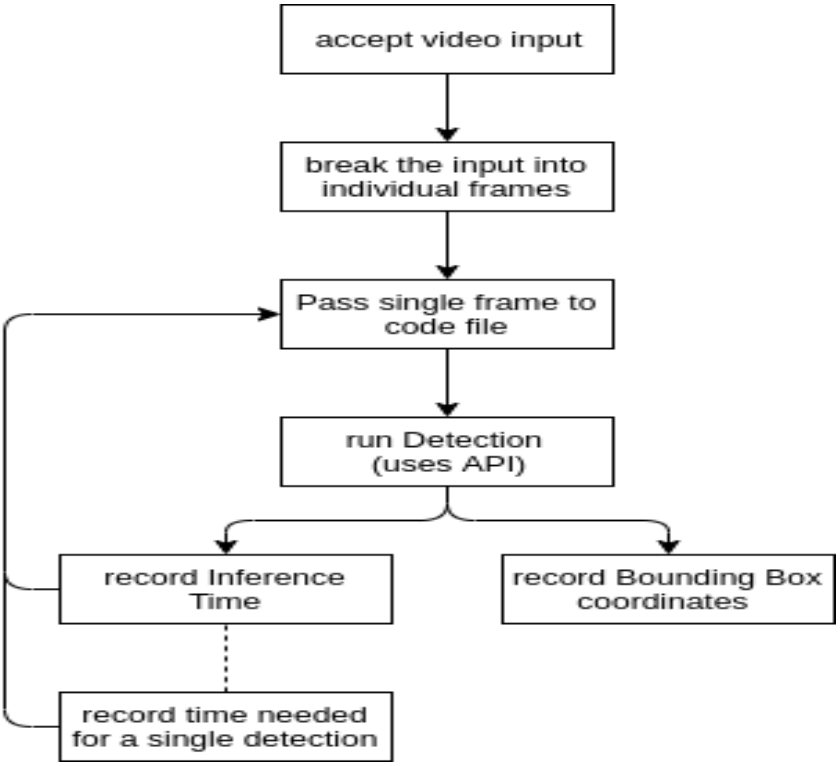


Figure 4.4: Working of the *Framewise* approach code file

4.1.2 Coupled (Detect and Track) approach

Similar to the previous approach, the video is treated in a frame-wise manner. However, one crucial difference in this approach is that two processes namely **Detection** and **Tracking** are implemented in tandem. The output of the **Detection** stage would be fed to the **Tracking** stage. Doing so would propagate the temporal information in the form of the previous location of the object down the pipeline. The object detector network used is **SSD MobileNet[35]** architecture (available in TensorFlow API) and the tracker algorithm is **Kalman filter** [28]. Open Loop Kalman Filter discussed in **Theoretical Background** would be implemented. This variant of filter does not implement the update step rather takes the predictions to be the output of the filter itself. It might be insightful to explore this version of filter as instead of making use of the update step, the deep learning frameworks output will be used to rectify the tracker if it starts drifting. Functionality

provided by **pykalman**[2] has been used in conjunction with the object detector. Modifications have been made to the TensorFlow API in order to make the **Kalman filter** compatible with **SSD Mobilenet** architecture. Certain frames (key frames) would be processed by the detector network and the remaining intermediate frames (non key frames) would be processed by the tracking algorithm. The concept of key frames and non-key frames can be visualised in Figure 4.5. Key frames are highlighted in green whereas the non key frames are white. The switching from **Detection** to **Tracking** and back again would be governed by a policy which has been devised after certain trials. First 5 frames of the input video and thereupon every 10^{th} frame is treated as a key frame. All the intervening frames are treated as non-key frames.

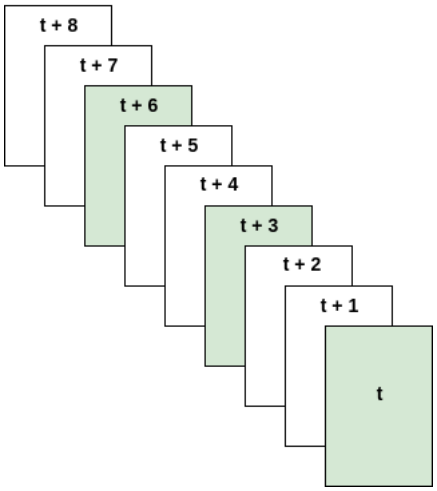


Figure 4.5: Detection and Tracking work in tandem

Figure 4.6 provides an overview of the workings of the approach:

- The key frames are provided as input (a) and the non-key frames are used as input for the (b)
- Output obtained from the key frames is the co-ordinates of the bounding box (this is the propagation of **temporal information** between subsequent frames), which are also fed to (b). The first 5 frames would be processed by (a) and afterwards till the 10^{th} frame, (b) processes the non-key frames along with the input from (a) and localize the object.
- (b) uses Kalman Filter to localize the object until either of the two conditions are met:
    1. next key frame occurs, in which case (a) is invoked again
    2. the tracker drifts substantially from the last detection whereupon the last stable detection output would be used as a rectification measure. The threshold for drifting has been incorporated in the code file. The threshold was established iteratively after visualizing the trials.

[2]https://www.hdm-stuttgart.de/ maucher/Python/ComputerVision/html/Tracking.html#kalman-filter

- For a single frame either (c) or (d) would be invoked. Once the processing of a single frame is completed, the next frame is checked whether it is a key frame or a non-key frame and accordingly (a) or (b) is called.

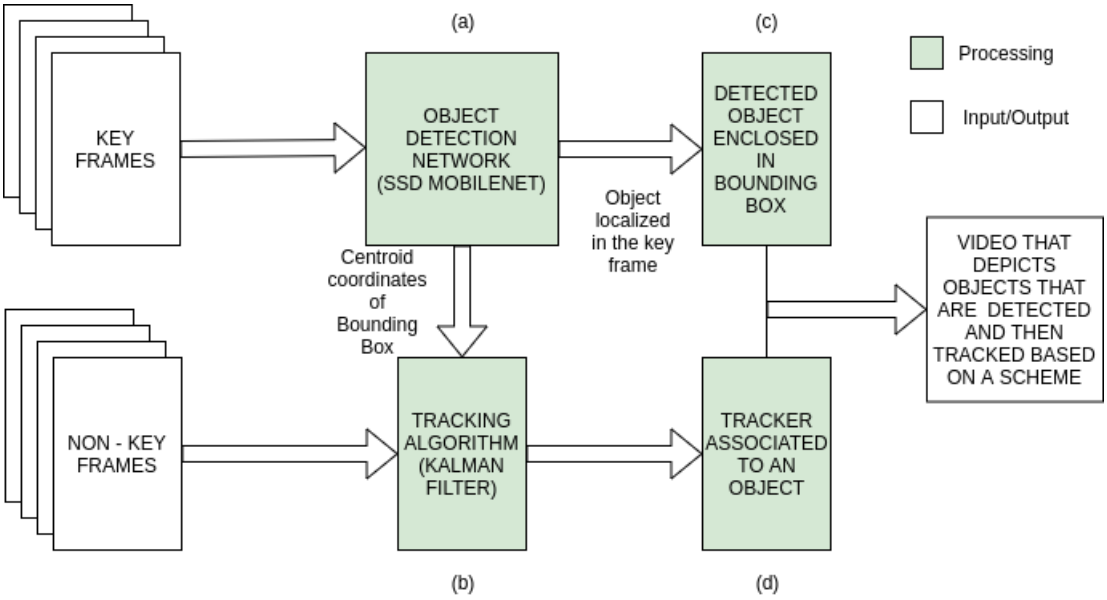


Figure 4.6: Pipeline adopted for *Coupled* approach

Figure 4.7 explains the coupling between (a) and (b) in Figure 4.6. It also provides a detailed insight into how the code handles the iteration of the aforementioned procedure.

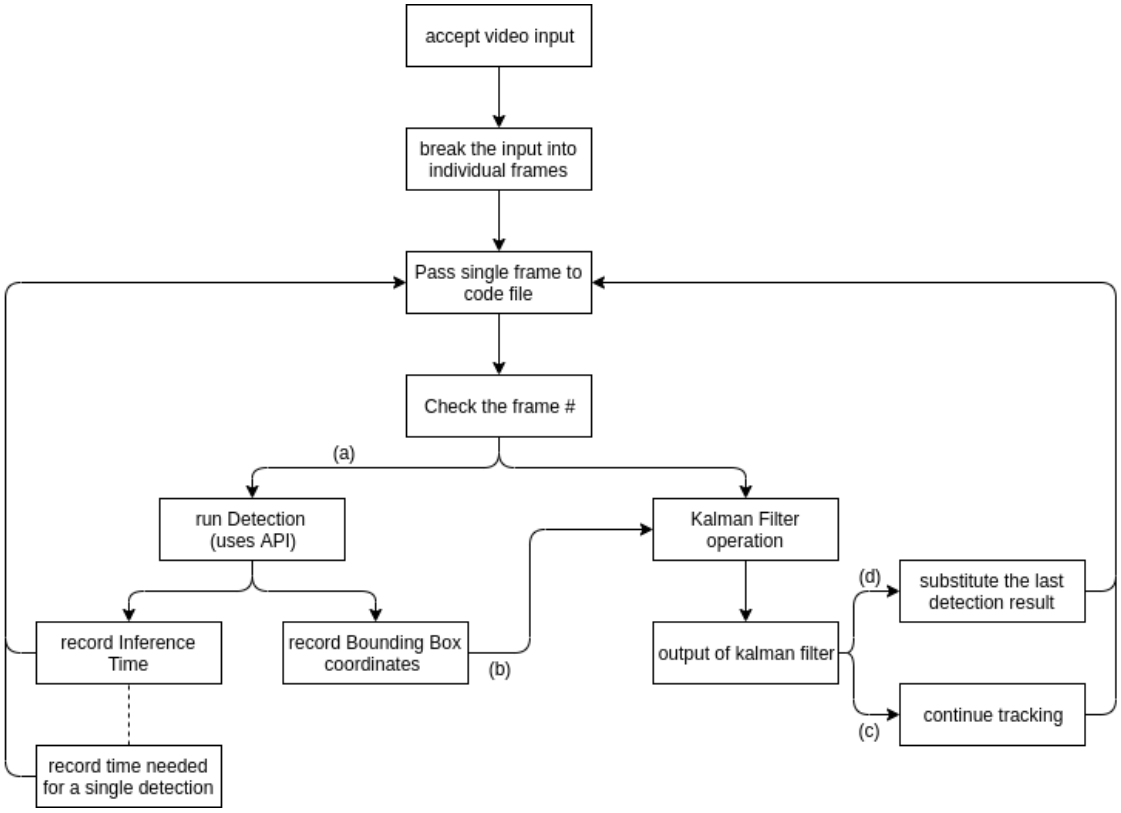


Figure 4.7: Working of *Coupled* approach code file
(a) first 5 frames and thereupon every 10th frame (check if key frame)
(b) last detection results just before switching the operation passed to tracker
(c) kalman filter output - last detection result < threshold
(d) kalman filter output - last detection result > threshold

4.2 Selection of comparison parameters

The last section discussed approaches and their pipelines. This section describes parameters used for comparison of the performance of the described approaches. These parameters would be helpful in quantifying the performances.

4.2.1 Intersection over Union (IoU)

Intersection over Union (IoU) is a parameter that evaluates the overlap between two boxes. In this work, we calculate the IoU between a Ground Truth bounding box and an Output bounding box. Ground Truth bounding box is obtained by manual annotations and the Output bounding box is obtained from the outputs of the approaches. For a perfect overlap, IoU equals 1 and for no overlap IoU equals 0. In this way, IoU will be used for quantifying the localization ability of the approach. The concept of IoU is better visualized by Figure 4.8.

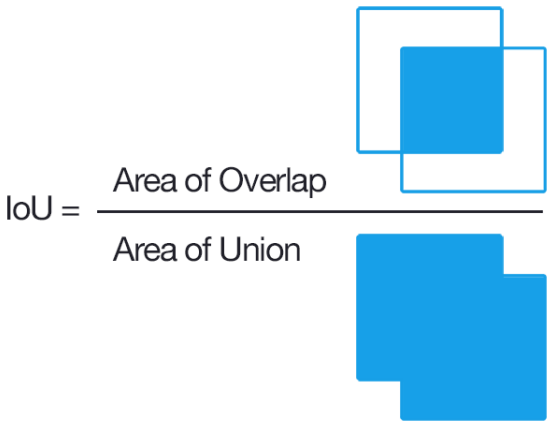


Figure 4.8: Intersection over Union[3]

4.2.2 Inference Time

Along with the localization ability of the approach, it is also crucial that the method should be able to run in less amount of time. In order to quantify the ability of the approach to run in less time, Inference Time is measured. It is essentially the amount of time taken by the approach to process the input and yield the output. Inference Time can be employed for analysis in two variants:

- **End to End**: The analysis and discussions conducted in this work considers this variant of the parameter. It is measured from the instant when the first frame is fed as input to the

[3]Image taken from: https://en.wikipedia.org/wiki/Jaccard_index

approach to the instant when the approach finishes processing the last frame of input video. This variant would be useful to calculate the number of frames per second (fps) as well.

- **Operation specific**: This is measured on a frame to frame basis. The measurement is based on how much time the approach takes to process a single frame. For the case of *Coupled* approach, it can be defined as the time taken by a single Detect or single Track operation. The results for this variant are provided in **Appendix B**. Since, the timing for each operations are recorded using this variant, detailed studies can be conducted in order to devise an adaptive Detecting-Tracking switching policy.

4.3 Test Data Specifications

In order to test the performances of the selected approaches, a number of conditions were incorporated while recording the test data (videos) to make the comparison more holistic. Each video can be considered to be a separate test case in itself. The objective while recording the videos was to cover those situations/conditions which might be encountered in real life. This section discusses these conditions in detail.

4.3.1 Recording Hardware

The primary consideration while making this decision was: does the resolution of the recording hardware affects the performance drastically or not. For this, two hardware devices were considered for recording the videos: *Mobile camera* and *USB camera*. The *Mobile camera* records videos at a resolution of 720×480 and the *USB camera* records at 640×480.

4.3.2 Object categories under consideration

The decision to select object categories for consideration was influenced by the choice of pre trained weight file used. Since this work makes use of **SSD MobileNet** which has been trained on MS COCO dataset, it became imperative to provide preference to object categories included in it. Two object categories *Cup* and *Scissors* were selected for recording the videos after practical considerations which are listed as follows:

- These two categories provided drastic variation in feature space. On one hand *Cup* consists of a cylindrical main body with a curvilinear handle whereas the *Scissor* consists of tapering main body with circular handles.

- These were the easiest object categories to work and record the test data with making the experiments easy to replicate. These object categories could be subjected to a variety of testing conditions and scenarios.

4.3.3 Recording Scenarios

Different scenarios were considered while recording the videos in order to test the versatility of the approaches. They were selected so as to replicate the real world conditions. These were as follows:

- *Standard* scenario was used to test the working of approaches in ideal conditions. While recording, the background was static and no other objects came in the scope of the camera lens. Since it tried to replicate ideal conditions, this scenario disregarded the occlusions as well.
- *Cluttered* scenario differed from the previous scenario in that it included other objects in the scope of camera lens along with the object under consideration. A desk setting was considered which had static objects. However, this scenario still did not incorporate occlusion conditions. Including the objects which were not of interest was useful for testing the localization ability of the approaches among the background clutter.
- *Occlusion* scenario included the conditions where the view of the object was momentarily obstructed by some other object. One point to consider is that the background was uncluttered as was in the case of *Standard* scenario.

From the aforementioned discussion, the total number of test cases generated can be calculated as follows:

- Two types of recording Hardware (*Mobile camera* and *USB camera*)
- Two types/categories of objects (*Cup* and *Scissor*)
- Three video recording scenarios (*Standard*, *Cluttered* and *Occlusion*)

The total number of test conditions generated by calculating the combinations equal $2 \times 2 \times 3$ i.e. 12 test cases.

4.4 Hardware

- CPU (Intel Core i5 8^{th} generation)
- USB Camera (640×480)
- Mobile Camera (720×480)

4.5 Software

- NumPy (1.16.4)
- OpenCV (4.1.0)

- Python (3.7.3)
- TensorFlow (1.14.0)
- pykalman (0.9.2)
- Matplotlib (3.1.1)

4.6 Assumptions

This section discusses the assumptions which have been considered while recording the videos and conducting the comparison of the approaches.

- The approaches have been evaluated on their ability to identify a single object from video despite presence of other objects in the vicinity.
- Random motion of either recording device or the object has been avoided.
- Ground Truth annotations exhibit consistency throughout and any minor errors in annotating are ignored.
- Video are recorded by moving the recording device and keeping the object stationary. By the principles of relative velocity, the opposite recording conditions is also assumed to work properly.
- Object class before entering the occlusion state and after exiting the occlusion state should be the same.

5

Results and Discussions

This chapter discusses the results of the implementation conducted for the comparative evaluation of the approaches discussed in the **Methodology**. *Framewise* and *Coupled (Detect and Track)* approaches were compared based on certain test conditions. A detailed discussion of the compared approaches and the test cases considered is provided in **Methodology**. A brief recap of the test cases has been provided here:

- two hardware devices were used for recording the videos (*Mobile Camera* and *Camera USB*) to check the robustness of detection against real world inputs.
- two classes of objects were considered (*Cup* and *Scissor*).
- three scenarios were considered for recording the videos (*Standard*, *Cluttered* and *Occlusion*).

Figure 5.1 provides a convention which can be used to understand the test cases. Detailed graphical results for the experimentation have been provided in the **Appendix**. However, the theoretical analysis and summarised results can be found in the current chapter.

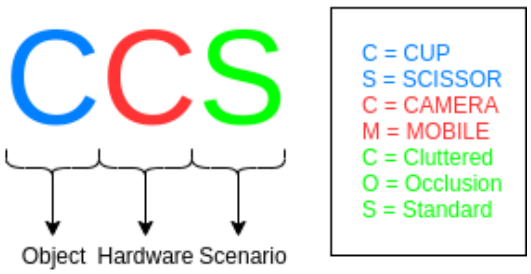


Figure 5.1: Convention

5.1 The Standard Scenario

The background in this scenario is uncluttered and the object suffers no occlusion. Figure 5.2 and Figure 5.3 depict the output for both approaches respectively.

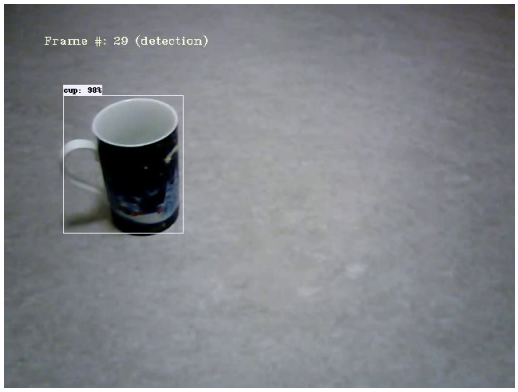


Figure 5.2:
Framewise CCS
Frame#29 (detection)

Figure 5.3:
Coupled CCS
Frame#13 (original tracker)

The performances for both the approaches on *Standard* scenario have been listed in Table 5.1. The parameters recorded are IoU and Inference Time (also in terms of frames per second). Following are the observations derived from Table 5.1:

- *Framewise* approach performs marginally better for two cases (CCS and CMS) than *Coupled* approach for IoU measurements. With the same performance metric, in the other two cases (SCS and SMS), both the approaches perform equally well.
- The comparison based on Inference Time is not as straightforward. *Coupled* approach performs better in two (CMS and SMS) cases whereas *Framewise* approach works better in remaining two cases (CCS and SCS).

Test Case	**Framewise**			**Coupled**		
	Avg. IoU	**Inference Time (ms)**	**Inference Speed (fps)**	**Avg. IoU**	**Inference Time (ms)**	**Inference Speed (fps)**
CCS	0.88	6.87	14.6	0.86	7.18	13.9
CMS	0.765	7.01	14.3	0.75	6.37	15.7
SCS	0.88	7.14	14.0	0.88	7.68	13.0
SMS	0.90	6.79	14.7	0.90	6.58	15.2

Table 5.1: Comparison of IoU and Inference Time performances for Standard scenario

5.2 The Cluttered Scenario

The background in this situation is cluttered and the object suffers no occlusion as can be seen in Figure 5.5 and Figure 5.4. Objects other than those of interest also occupy the frame. The

output for *Framewise* and *Coupled* approach have been shown in Figure 5.4 and Figure 5.5.

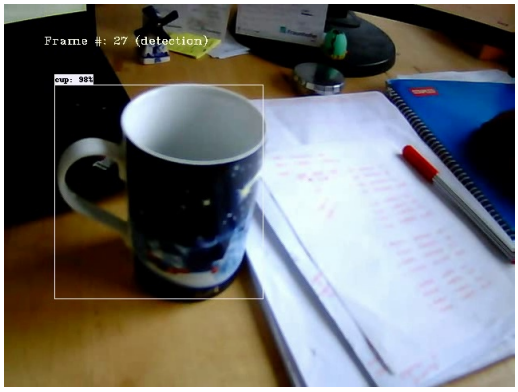


Figure 5.4:
Framewise CCC
Frame#27 (detection)

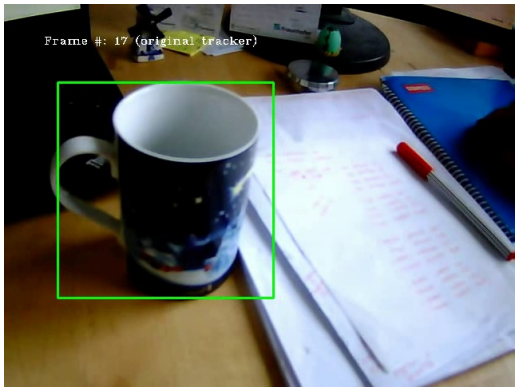


Figure 5.5:
Coupled CCC
Frame#17 (original tracker)

Table 5.2 provides a summary of the performances of both the approaches when compared in terms of IoU and Inference Time.

Some important observations for *Cluttered* scenario are:

- *Coupled* approach performs slightly worse than *Framewise* for all cases in this scenario when compared in terms of IoU measurements.
- However, when comparing the Inference Time (& subsequently the frames per second), it is observed that the *Coupled* approach performs better than the *Framewise* approach in all the cases.

Test Case	**Framewise**			**Coupled**		
	Avg. IoU	**Inference Time (ms)**	**Inference Speed (fps)**	**Avg. IoU**	**Inference Time (ms)**	**Inference Speed (fps)**
CCC	0.87	7.86	12.7	0.85	7.09	14.1
CMC	0.895	6.71	14.9	0.885	6.08	16.4
SCC	0.83	7.47	13.4	0.76	7.33	13.6
SMC	0.86	7.12	14.0	0.85	6.12	16.3

Table 5.2: Comparison of IoU and Inference Time performances for Cluttered scenario

5.3 The Occluded Scenario

The background in this situation is uncluttered and the object suffers occlusion unlike in the previous scenarios. This can be observed in Figure 5.6 and Figure 5.7. These figures are also the output of approaches in this scenario.

Figure 5.6:
Framewise CCO
Frame#84 (detection)

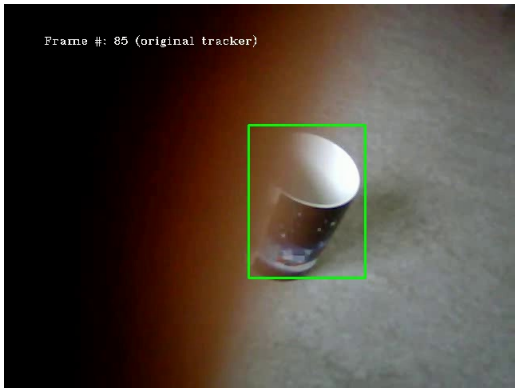


Figure 5.7:
Coupled CCC
Frame#17 (original tracker)

Figure 5.6 and Figure 5.7 depicts the beginning of occlusion event and eventually the object (in several succeeding frames in this video) will undergo occlusion.

Table 5.3 provides the IoU and Inference Time performances of both the approaches. Following points can be summarized:

- *Coupled* approach works better for the two cases (CCO and CMO). However, for the remaining two cases (SCO and SMO), *Framewise* approach works better tha *Coupled* approach

- Considering the Inference Time, *Coupled* approach performs better than *Framewise* approach for two of the cases (CMO and SMO). *Framewise* works marginally better for the other two (CCO and SCO)

Test Case	**Framewise**			**Coupled**		
	Avg. IoU	**Inference Time (ms)**	**Inference Speed (fps)**	**Avg. IoU**	**Inference Time (ms)**	**Inference Speed (fps)**
CCO	0.61	6.92	14.5	0.62	7.07	14.2
CMO	0.76	7.13	14.0	0.78	6.72	14.9
SCO	0.72	6.83	14.6	0.695	7.81	12.8
SMO	0.76	6.93	14.4	0.73	6.30	15.9

Table 5.3: Comparison of IoU and Inference Time performances for Occluded scenario

The occlusion scenario presented interesting results as can be seen in Figure 5.8 and Figure 5.9. These figures depict the variation of IoU score over a course of 100 frames. In both the figures, it can be observed that occlusion affects *Framewise* approach more than it does *Coupled* approach. The only instance where *Coupled* approach would be affected by occlusion is the instance when the Detection operation is to be conducted but the object is not visible.

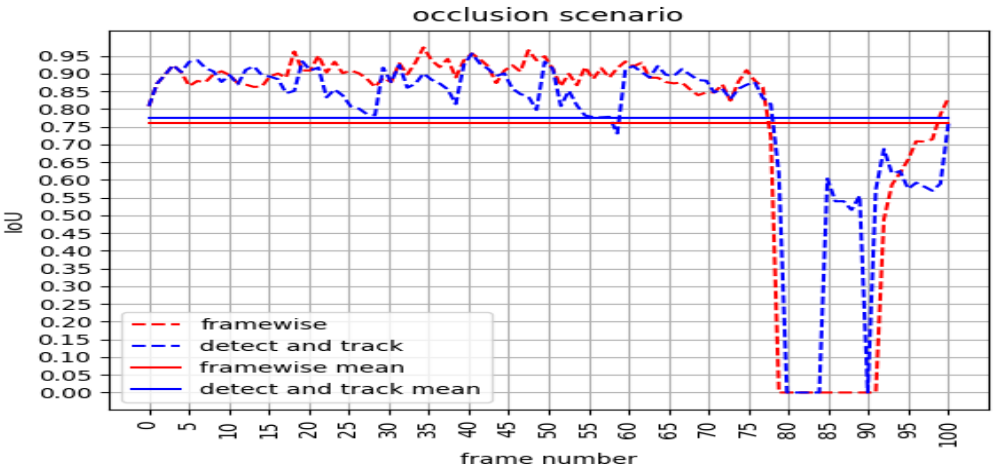


Figure 5.8: CMO IoU variation

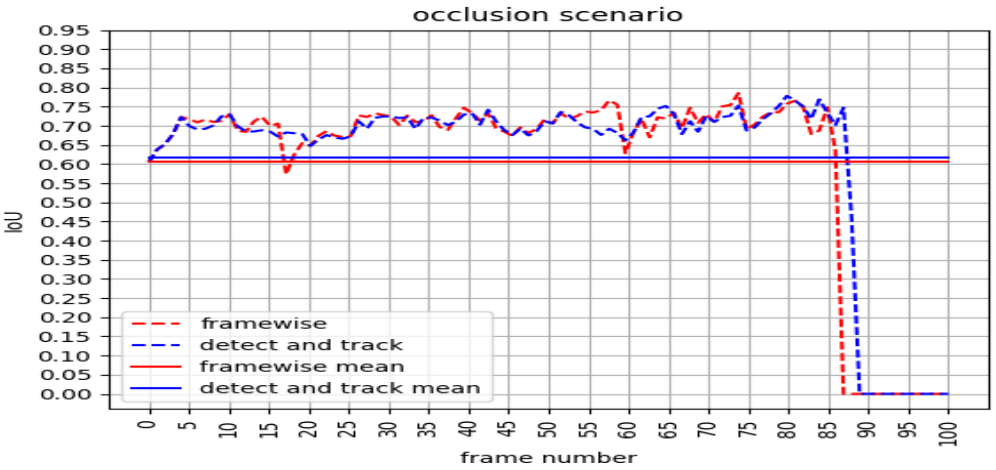


Figure 5.9: CCO IoU variation

5.4 Summarized Results and Discussions

This section provides an overview and a qualitative as well as quantitative analysis of the results. Based on Table 5.1, Table 5.2 and Table 5.3, a summarized visualization of the results has been provided in Figures 5.10 and 5.11.

5.4.1 Qualitative Analysis

Intersection over Union

Figure 5.10 visualizes the average Intersection over Union values for both *Framewise* and *Coupled* approach and all the test cases.

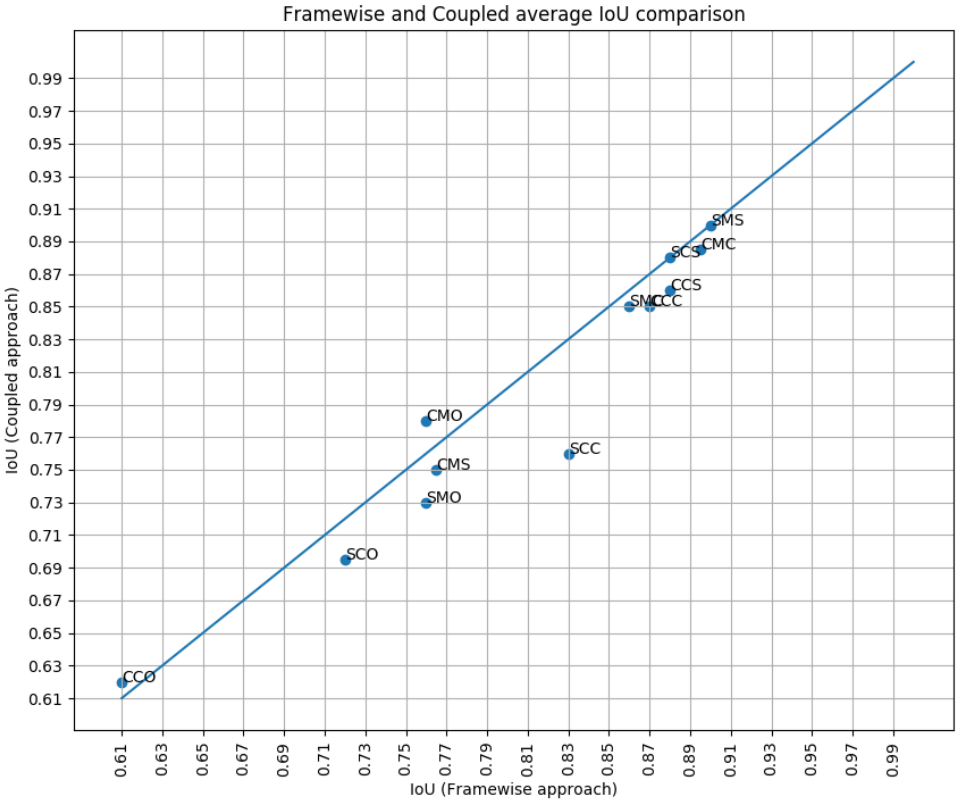


Figure 5.10: Average IoU results for all test cases

Each point in Figure 5.10 denotes a test case listed in Table 5.1, Table 5.2 and Table 5.3. The line (y=x) is the visualization of equal performances of both the approaches as any point (test case) lying on this line implies that the particular test case has the same value of Intersection over Union for both the approaches. Points lying below the line mean that those test cases have higher IoU for *Framewise* approach whereas those lying above would mean the *Coupled* approach has better IoU performances. One interesting observation here is that farther away the point is from the line (y=x), more the disparity in performance of the approaches. If the point is farther below the line then *Framewise* performs substantially better than *Coupled* approach. Similarly, if the point lies farther above the line then *Coupled* approach performs considerably better than *Framewise* approach. This can be understood easily with an example. For instance, in Figure 5.10, the point (test case) **SCC (Scissor Camera Cluttered)** is farther below the line (y=x). This implies that for this test case the performance for *Framewise* approach is markedly better than *Coupled* approach. Based on this understanding of the graphs, following are the observations from Figure 5.10:

- *Coupled* approach performs better for CMO and CCO.
- *Framewise* approach performs better for CCS, CMS, CCC, CMC, SCC, SMC, SCO and SMO.
- Both approaches perform equally well for SMS and SCS.

Inference Time

Figure 5.11 depicts the average Inference Time (end to end) for the *Framewise* and *Coupled* approaches.

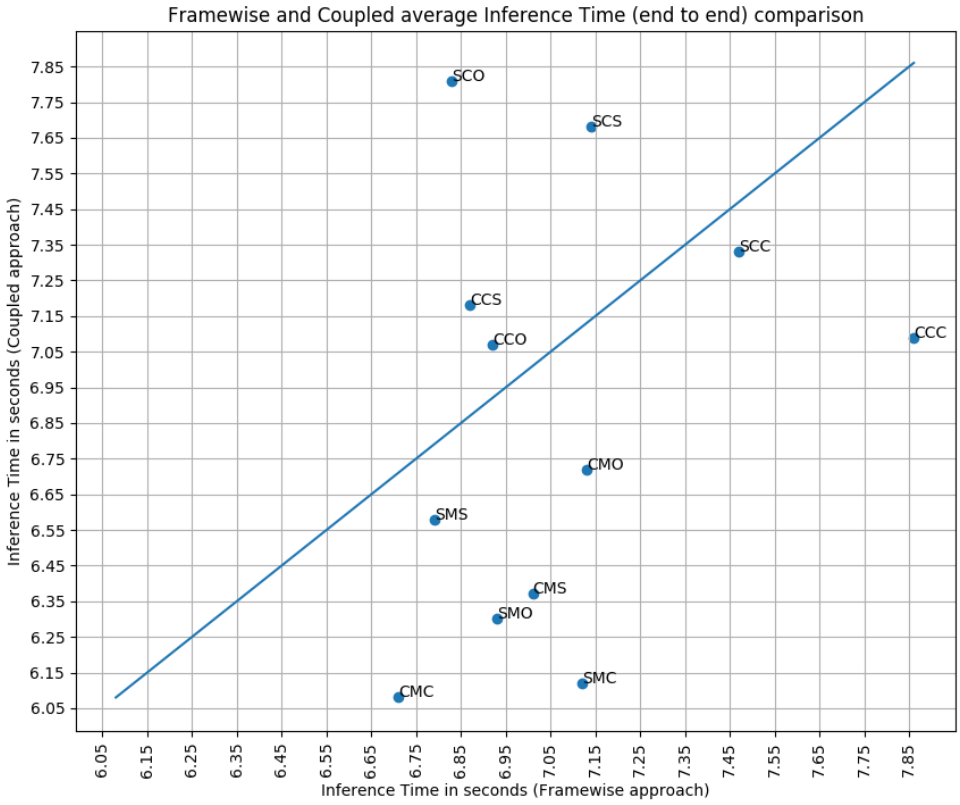


Figure 5.11: Average Inference Time results for all test cases

Similar to previous section, the points in Figure 5.11 depict test cases listed in Table 5.1, Table 5.2 and Table 5.3. Just as in the previous section, the line (y=x) is the visualization of equal performance of both the approaches and any point (test case) lying on this line implies that the time taken by both approaches to process would be same. However, in this case the understanding of the graph would be opposite to that shown in Figure 5.10. Points (test cases) lying below the

line would mean that the *Coupled* approach takes lesser time to run and hence are quicker (yielding higher frame per second) whereas the points lying above the line would mean that *Framewise* approach is quicker and yields a higher frames per second count. Also, points lying farther below the line would mean that the time taken by *Coupled* approach is considerably lesser than *Framewise* approach. Points lying farther above the line would mean that the time taken by *Coupled* approach is considerably more than *Framewise* approach. This can be easily understood by an example. In Figure 5.11, the point **CCC (Cup Camera Cluttered)** is farther below the line, it means that for this test case the *Framewise* approach takes more time than *Coupled* approach.

From Figure 5.10, following are the observations:

- *Coupled* approach performs better for CMC, SMS, SMO, CMO, CMS, SMC, SCC and CCC
- *Framewise* approach performs better for SCO, SCS, CCS and CCO

5.4.2 Quantitative Analysis

This section analyses the results on a quantitative basis. The results have been condensed in a tabular form. Table 5.4 is obtained by combining the results from Table 5.1, Table 5.2 and Table 5.3. It provides the percentage variation between performances of both the approaches for IoU and Inference Time.

In Table 5.4, the columns represent the percent change of values in the performances for IoU and Inference Time for both the approaches. As can be seen, the values are calculated as percent variation of *Coupled* approach with respect to *Framewise* approach. The positive percentage values of IoU and negative percentage values of Inference Time (and hence positive percentage values of Inference Speed) would indicate that *Coupled* approach performs better than *Framewise* approach. Such occurrences are highlighted in the Table 5.4 for a quick overview. Calculating the percentage variation of parameters in one approach with respect to other approach facilitates the comparison of the approaches on grounds of parameters. Following are the observations which can be made on the basis of Table 5.4:

Intersection over Union

In terms of IoU, *Framewise* performs better in all cases except for four (CCO, CMO, SCS and SMS). Two of these cases (SCS and SMS) exhibit equal performances of IoU for both approaches. Rest of the two cases (CCO and CMO) see *Coupled* approach perform better. In the case of SCC the *Coupled* approach is comprehensively outperformed (by a margin of 8.4%) by *Framewise* approach as can be seen in Table 5.4. The poor performance of *Coupled* approach can be attributed to the fact that tracker might suffer drift over a prolonged period of time hence directly affecting the localization ability of the approach.

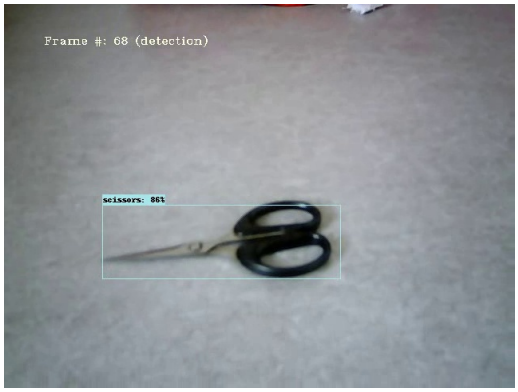


Figure 5.12:
Better localization in Framewise output
Frame#68 (detection)

Figure 5.13:
Tracker drift in Coupled output
Frame#68 (original tracker)

In Figure 5.13, the tracker has been operational for 8^{th} consecutive frame, this has led to it drifting from the actual object location. This can be seen in Figure 5.12 and Figure 5.13. The *Framewise* approach is able to correctly localize the object but *Coupled* approach drifts substantially.

An interesting case to note is scenario of occlusion, IoU for *Coupled* approach are higher (for two test cases) than that of *Framewise* approach by an order of 1-2%.

Figure 5.14:
Framewise approach fails in case of occlusions
Frame#86 (detection)

Figure 5.15:
Coupled approach works during occlusions
Frame#87 (original tracker)

This happens because the *Framewise* approach fails to detect the object in cases of occlusion whereas the *Coupled* approach estimates the position based on Kalman tracker calculation and yields an estimated position rather than totally missing the object's whereabouts.

This can be seen in Figure 5.14 and Figure 5.15, where the *Framewise* approach fails completely but *Coupled* approach works. This is despite the fact that a greater portion of object is hidden in case of *Coupled* approach when compared to *Framewise* approach. However, this explanation

doesn't apply to two occlusion cases where *Framewise* approach outperforms *Coupled* approach. On closer analysis, it is found that for SCO, the camera suffers abrupt motion just after occlusion event ends hence the tracker fails to localize the object correctly. This can be attributed to the fact that the tracker operates on a linear motion model (constant velocity) and random motions might result in erroneous calculations. In order to accommodate a random motion, a non linear model would be useful. As for the case of SMO (Scissor Mobile Occlusion), it is observed that due to occlusion at a crucial juncture (the frame where Tracking-Detection switch was about to take place), the object is not detected on this particular frame and the tracker is forced to work on old detection values and hence the drift in the tracker keeps on increasing.

Test Case	**Percent change in Coupled approach as compared to Framewise approach**		
	Average IoU (% change in ms)	**Inference Time (% change in ms)**	**Inference Speed (% change in fps)**
CCS	-2.27	4.51	-4.8
CMS	-1.96	-9.13	9.79
SCS	0	7.56	-7.14
SMS	0	-3.09	3.4
CCC	-2.29	-9.80	11.02
CMC	-1.12	-9.40	10.07
SCC	-8.4	-1.87	1.49
SMC	-1.16	-14.0	16.43
CCO	1.64	2.17	-2.06
CMO	2.63	-5.75	6.43
SCO	-3.47	14.34	-12.33
SMO	-3.9	-9.1	10.41

Table 5.4: Quantitative analysis (percent based) of IoU and Inference Time performances for all scenarios. Green highlights the instances when *Coupled* approach performs better than *Framewise* approach

Inference Time

In terms of Inference Time, *Coupled* approach performs better than *Framewise* approach in majority of the test cases. However, for four cases (CCS, SCS, CCO and SMO) the *Framewise* approach performs better. The primary reason as to why the *Coupled* approach performs better in terms of Inference Time can be analysed by looking at the graphs in **Appendix B**. The graphs depict the variations of time needed for operation at a specific frame. It can be seen that Kalman filter operations take significantly lesser amount of time to process a single frame than Detection operations do. This can be the reason as to why on a cumulative scale, *Coupled* approach takes lesser amount of time than *Framewise* approach as the number of Detection operations in the former are considerably lesser. However, this is contradicted by the four test cases as mentioned

earlier. The possible reason as to why these perform as exceptions may be attributed to irregular CPU load during their execution.

From the Table 5.4, it can be inferred that for minor decrements in IoU, *Coupled* approach yields substantially higher gains in Inference Time than *Framewise* approach. This can be attributed to the fact that Kalman filter operation take lesser time than a Detection operation.

For a detailed analysis of the operation specific inference time the corresponding plots have been provided in the **Appendix B**. These can be useful when the Detection-Tracking switching policy needs to be made more dynamic. Currently, the switching between the Detection and Tracking process (in *Coupled* approach) takes place every 10^{th} frame (after initial sustained spell of detection for 5 frames). This value of 10 frames was arrived by certain trials.

The figures attached in the previous sections depicted the situations being discussed. However, as can be seen in Figure 5.1, a total of two object categories and two recording devices were used to create the test cases. Owing to space limitations in the report, outputs from all the test cases have not been included here. These results have been added in **Appendix A and Appendix B**.

6

Conclusions and Future Scope

The previous chapter discussed the results obtained during the implementation phase of the project work. This chapter is an extension to the analysis and tries to draw some insights from the results as well as discuss the future scope of the work.

6.1 Conclusions

The section **Conclusion** has been further divided into two subsections namely **Survey** and **Implementation**.

6.1.1 Survey Conclusions

This section discusses the conclusions drawn from the analysis of papers conducted in the previous chapters. Following are the findings of the aforementioned analysis:

- Two primary approaches to conduct video object detection are:
  - Run object detector on individual frames repeatedly and hence not taking the advantage of additional information available in videos.
  - Include temporal information with object detector.
- The available approaches can be divided on the basis of following three categories in order to have a clear understanding:
  - Temporal Information Usage
  - Framewise Feature Extraction
  - Approach
- Certain methods were observed to be invariably referred to by other researchers. These methods [21], [30], [55], [54] and [56] have been referred to as **Contemporary Methods** and have been discussed in the chapter **State of the Art**.

6.1.2 Implementation Conclusions

This section discusses the benefits and drawbacks of the implemented approaches.

Pros (*Framewise* approach)

- The most intuitive method to process a video and detect objects in it is to take an individual frame and detect objects in it. *Framewise* approach exactly does this and its results can be considered as baseline to compare the performances of other approaches.
- High localization ability in Standard and Cluttered scenarios.
- No complicated switching policy as only a single operation (Detection) is iterated over the entire input.
- Independent of mathematical model and can handle random motion of object/camera.

Cons (*Framewise* approach)

- Poor object localization in prolonged occlusion scenarios as each Detection operation is completely independent of previous results and hence no temporal information is used.
- A single Detection operation takes longer than a Kalman filter operation.
- Issues in input videos such as loss of focus, lighting or awkward pose of object can cause Detection operation to miss the object entirely.
- Does not incorporate the temporal information aspect of the video.

Pros (*Coupled* approach)

- In this approach, the temporal information is being propagated in the form of previous detection results which are used in order to localize the object in the current frame.
- Kalman filter operation is relatively quicker than a Detection operation.
- Can effectively handle Occlusion scenario (even complete occlusion) as Detection information from past frames (temporal information) is handed on to subsequent frames.
- Can be made robust by incorporating additional mathematical models such as non linear motion model.
- Able to handle issues in input video granted that these do not occur on the frame designated for Detection operation.

Cons (*Coupled* approach)

- Currently, a rigid switching policy being used which designates the Detection frames and Tracking frames beforehand. This exposes the approach to failure because if the Detection operation is conducted on a frame when the input is difficult to process, the subsequent Tracking operations might also fail. Adaptive switching policy can avoid this.
- Mathematical model dependency, random motion of object/camera will yield erroneous results.
- With the current frame scheduling policy, if the object is not detected during the first few frames, this approach would fail as the tracker would not initialise.

Insights

From the aforementioned discussions, following conclusions can be drawn:

- If better localization is the requirement, *Framewise* approach would be advisable. *Coupled* approach can also be used however, an adaptive switching between Detection-Tracking needs to be planned to improve localization.
- If ability to handle random relative motion between the object and camera is desired, *Framewise* approach would be appropriate.
- If Inference Time (also Inference Speed) needs to be minimised, *Coupled* approach would be preferable.
- If ability to handle complete occlusion is desired, *Coupled* approach would be better choice.
- Approaches are hardware independent. As long as the detector network is able to extract features from the input and localize the object, both the approaches would perform and have no drastic variation in performances.
- Considerable gains in Inference Time for *Coupled* approach were observed over *Framewise* approach for marginal performance deterioration in Intersection over Union.

6.2 Future work

- The comparison can be made more extensive by extending it to multiple object classes rather than just two classes.
- Code optimization for Kalman filter and also possibly implementation for extended Kalman filter, unscented Kalman and other filters such as Particle filter can be done.
- In case of *Coupled* approach, instead of working with a rigid Detection-Tracking switching policy, an adaptive switching can be used.

- The comparison on the basis of other parameters such as mean Average Precision (mAP), Recall and Precision can be conducted which would provide a holistic perspective.
- Using a different base network/detection neural network could provide more information about how it performs on the parameters discussed in this work.
- Additional tests can be conducted by making use of hardware such as Time of Flight camera which provides depth information and the performances of the approach on this hardware can be gauged.
- Approaches employing RNNs or LSTMs can also be implemented to compare how they perform against the approaches implemented in this work.
- Towards the later stages of this work, a framework [38] was found which could be helpful to conduct the comparison in a more systematic manner.

A

Intersection over Union Graphs

This chapter contains the graphs for the test cases discussed in previous chapters for Intersection over Union between output of the approach and ground truth. These graphs are obtained using a script which has been discussed in **Appendix C**. Figure A.1 depicts the convention followed for the subsequent graphical results. Theoretical analysis of these results can be found in **Results**.

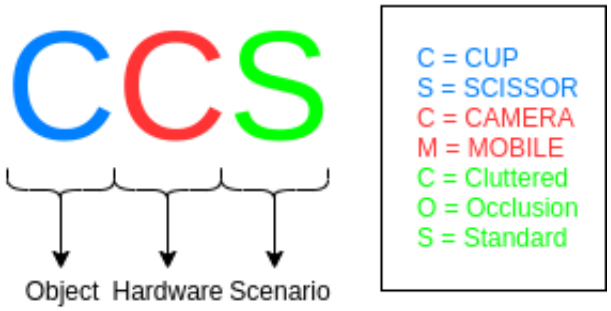


Figure A.1: Convention

A.1 The Standard Scenario

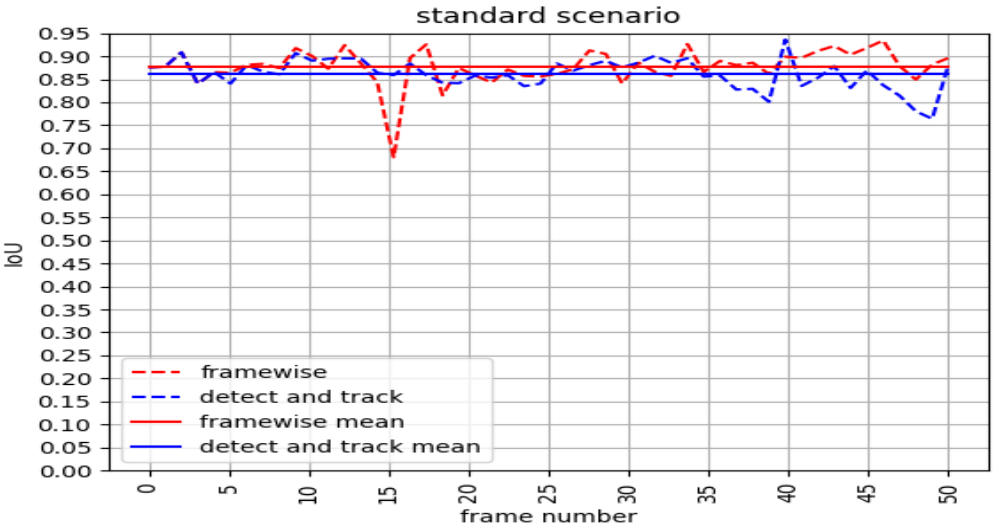
This section contains the graphical results from the *Standard* scenario. Figures A.2 and A.3 depict the results.

A.2 The Cluttered Scenario

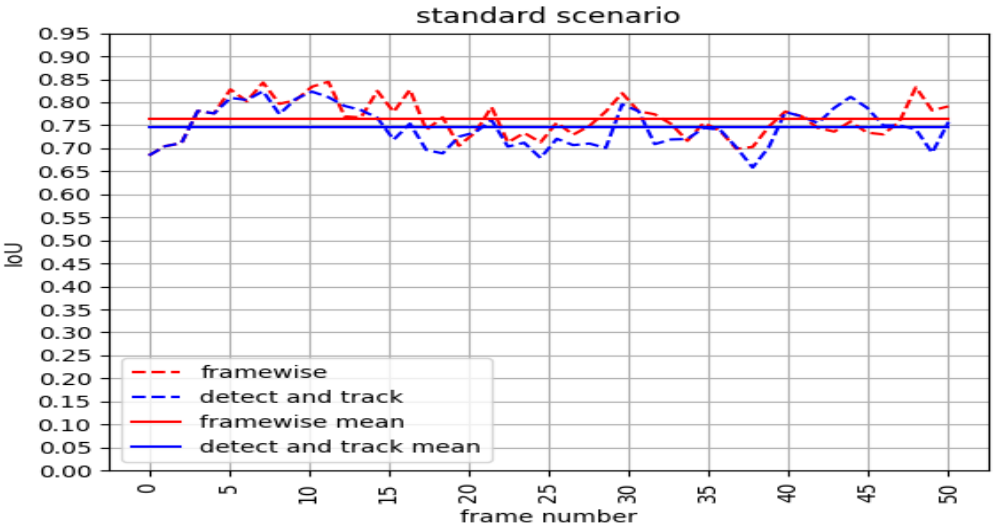
This section contains the graphical results from the *Cluttered* scenario. Figures A.4 and A.5 depict the results.

A.3 The Occlusion Scenario

This section contains the graphical results from the *Occlusion* scenario. Figures A.6 and A.7 depict the results.

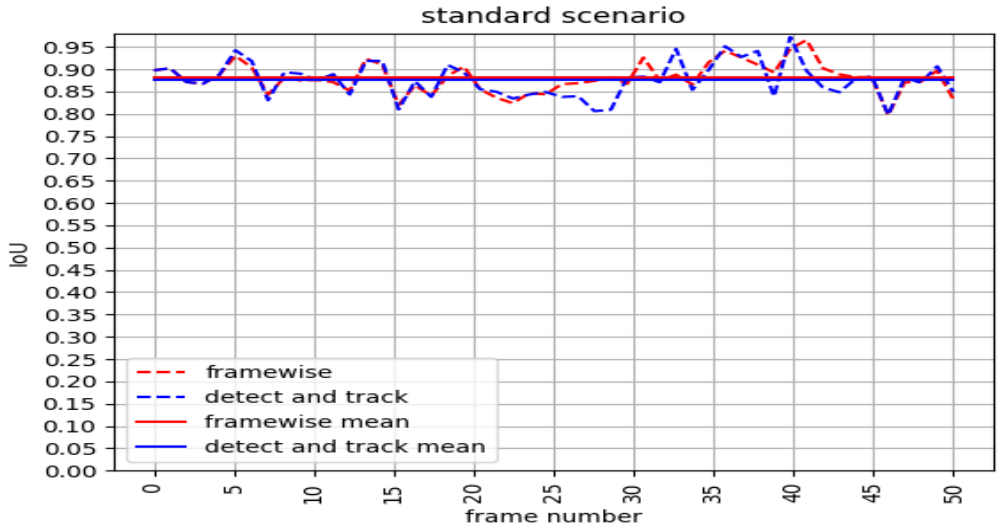


(a) CCS

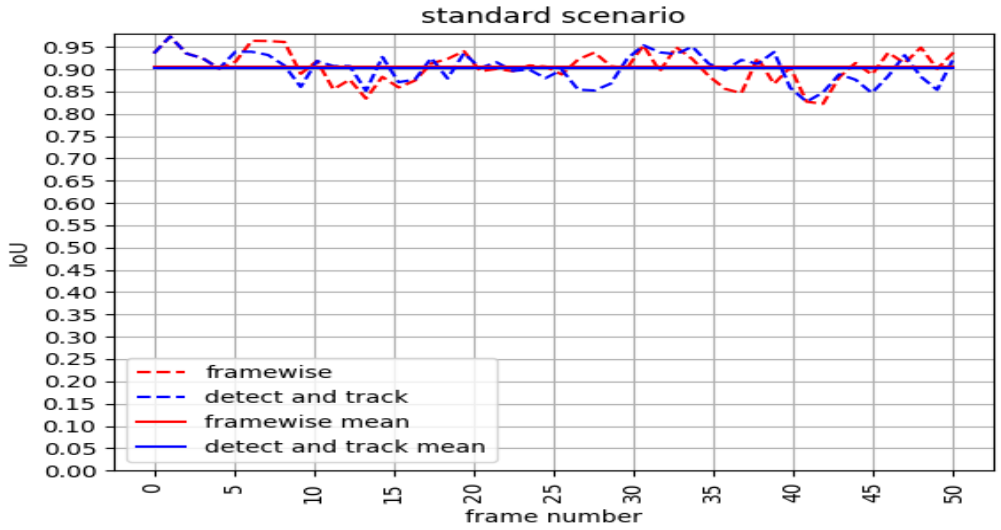


(b) CMS

Figure A.2: CCS and CMS Intersection over Union

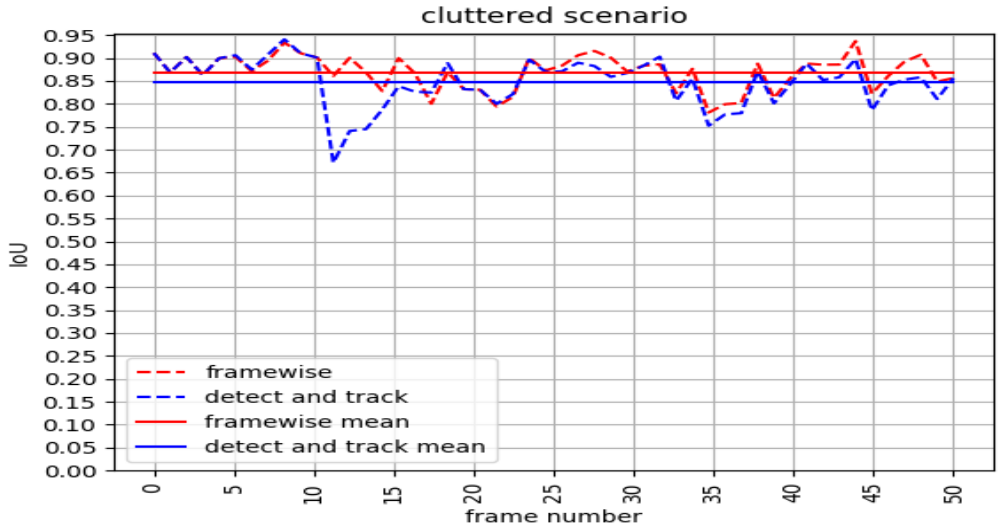


(a) SCS

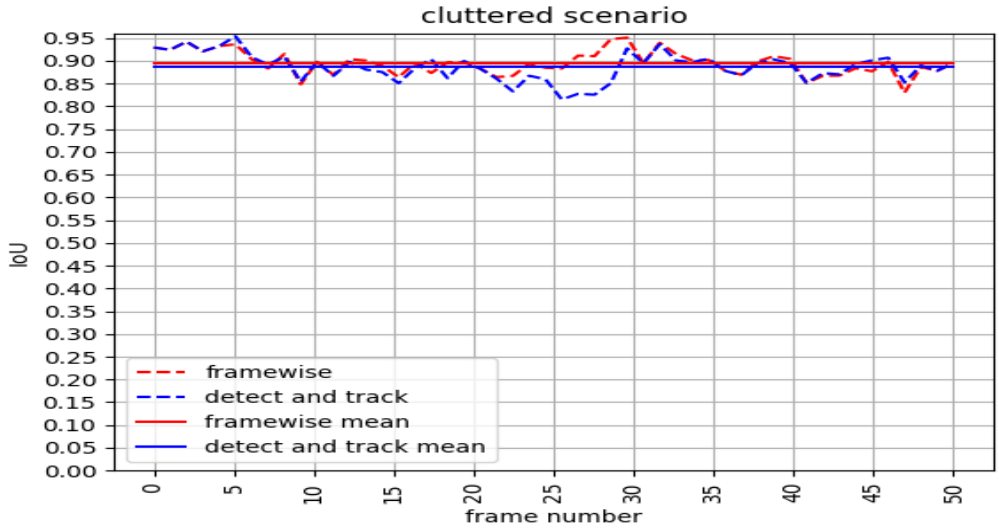


(b) SMS

Figure A.3: SCS and SMS Intersection over Union

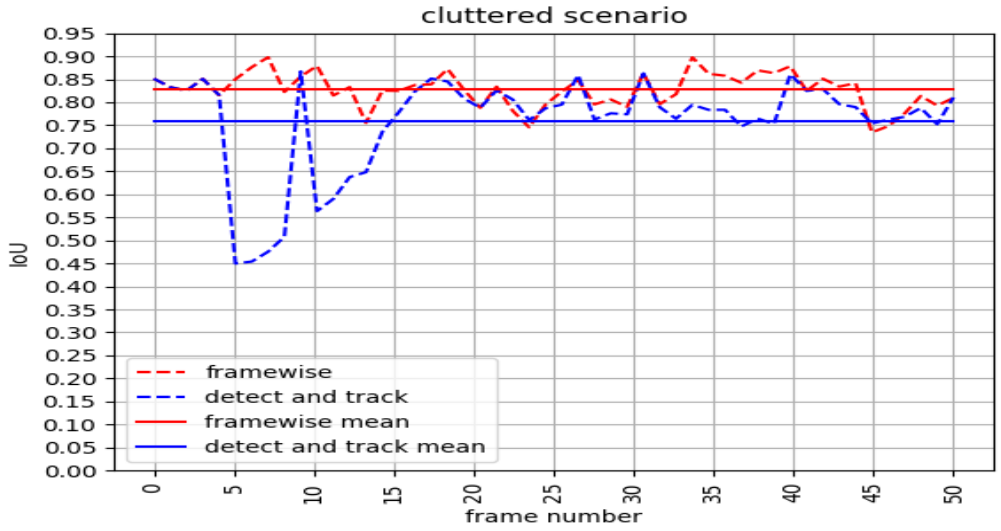


(a) CCC

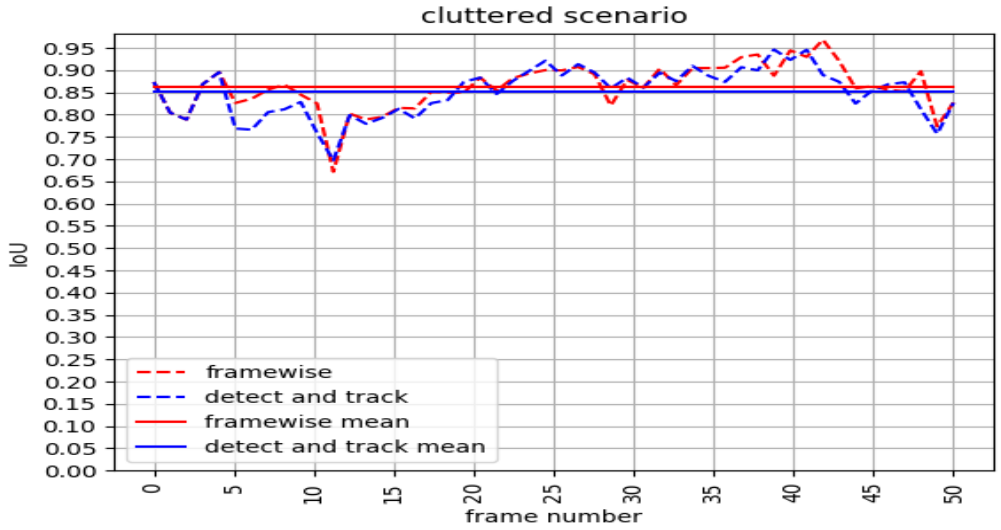


(b) CMC

Figure A.4: CCC and CMC Intersection over Union

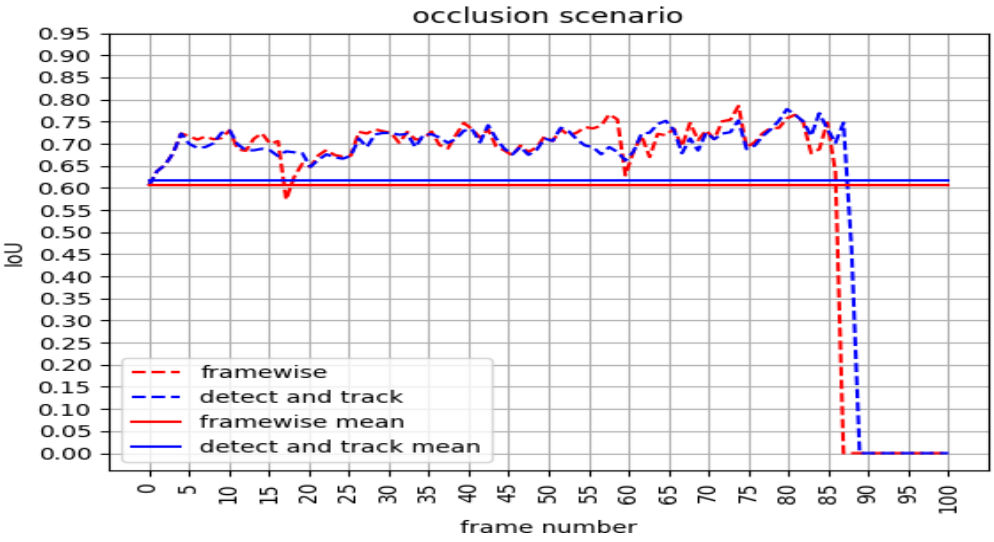


(a) SCC

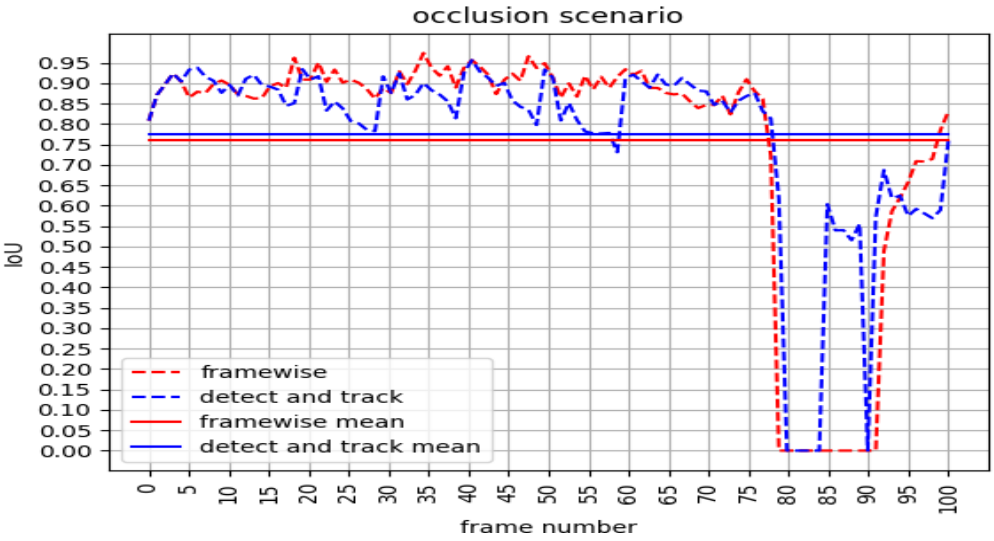


(b) SMC

Figure A.5: SCC and SMC Intersection over Union

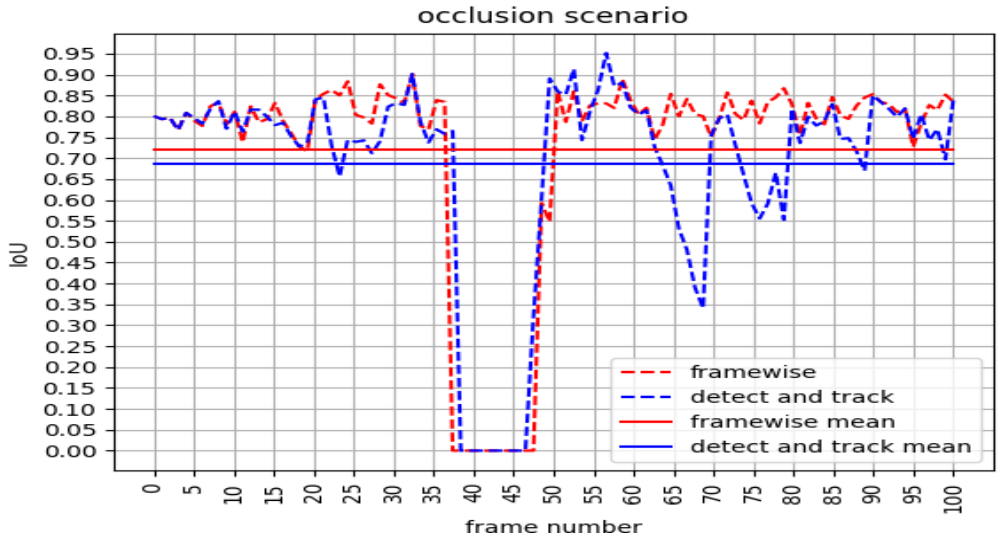


(a) CCO

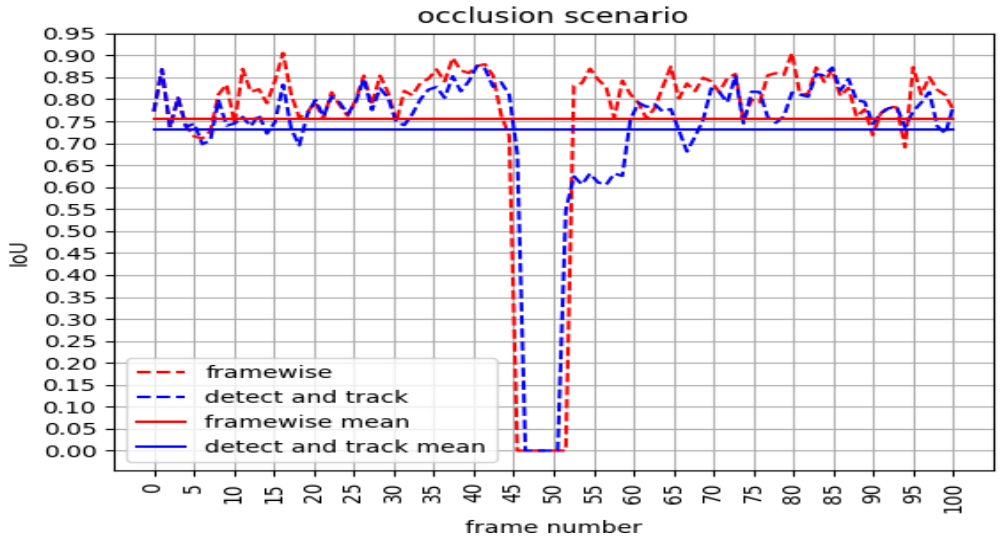


(b) CMO

Figure A.6: CCO and CMO Intersection over Union



(a) SCO



(b) SMO

Figure A.7: SCO and SMO Intersection over Union

B

Inference Time Graphs

This chapter contains the graphs for the test cases described in previous chapters. Inference time for each operation (Detection and Tracking) in both the approaches has been visualized and has been plotted by measuring timings for every single operation in the approach. The understanding of the graphs is as follows:

- For *Framewise* approach, timing for every detection operation has been measured and plotted against the corresponding frame number
- Similarly, for *Coupled* approach, timing for every detection as well as timing for every tracking operation has been measured and plotted against the corresponding frame number

This plot will help in analysing the scheduling policy of detection and tracking and come up with an adaptive switching policy. A similar convention to that of **Appendix A** followed for the subsequent graphical results.

B.1 The Standard Scenario

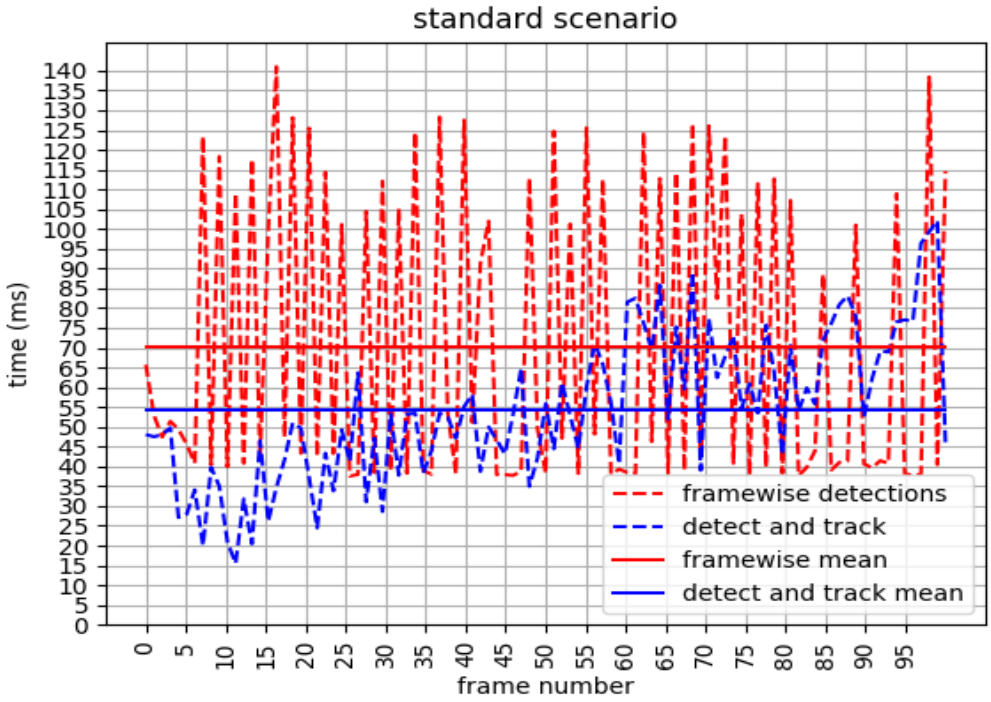
Figures B.1 and B.2 depict the results.

B.2 The Cluttered Scenario

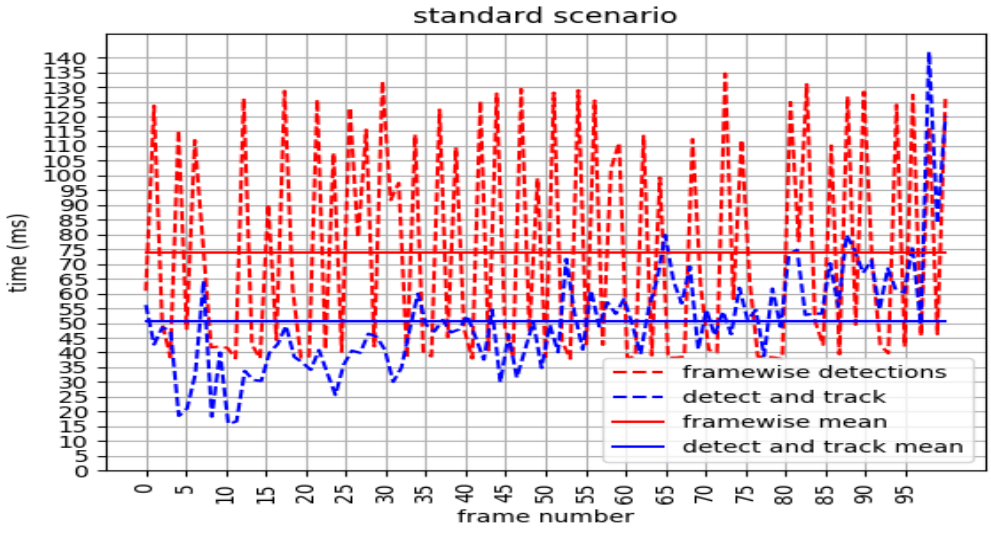
Figures B.3 and B.4 depict the results.

B.3 The Occlusion Scenario

Figures B.5 and B.6 depict the results.

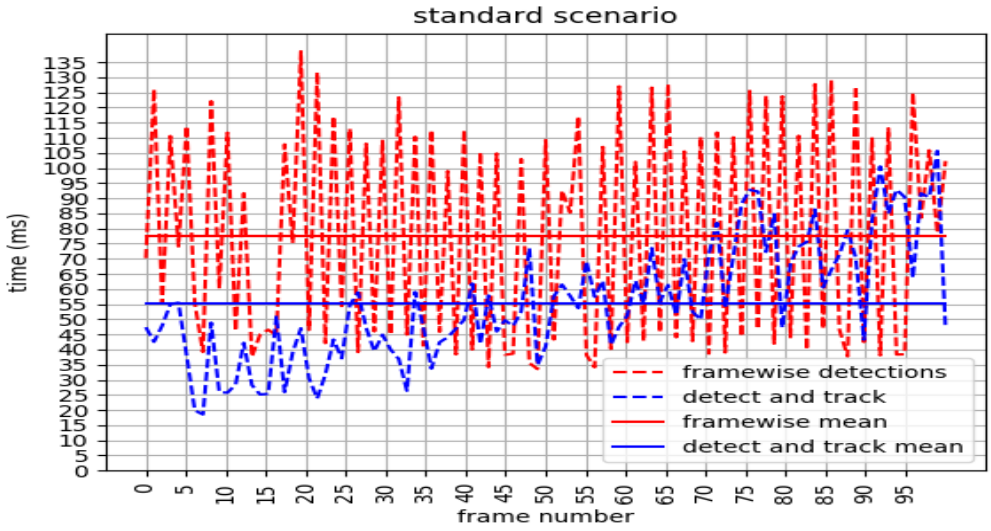


(a) CCS

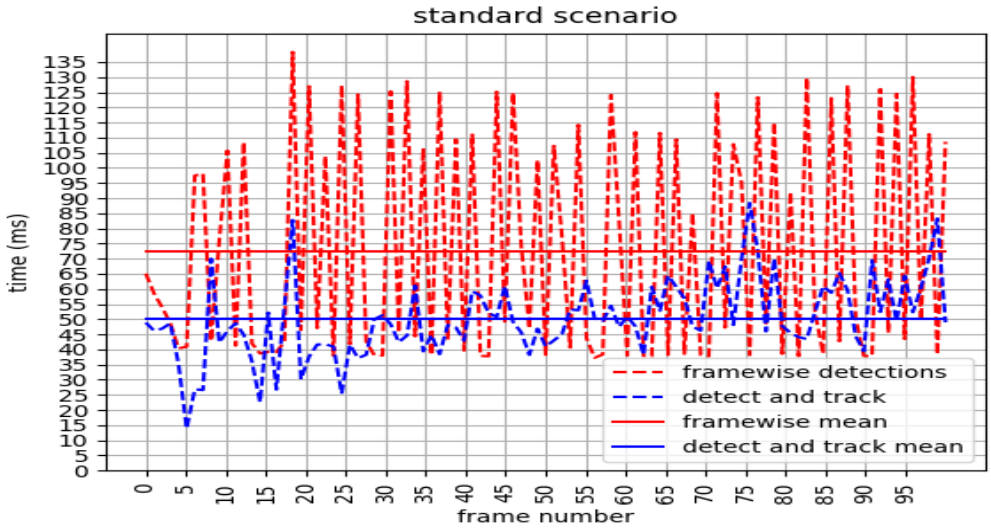


(b) CMS

Figure B.1: CCS and CMS Inference Time

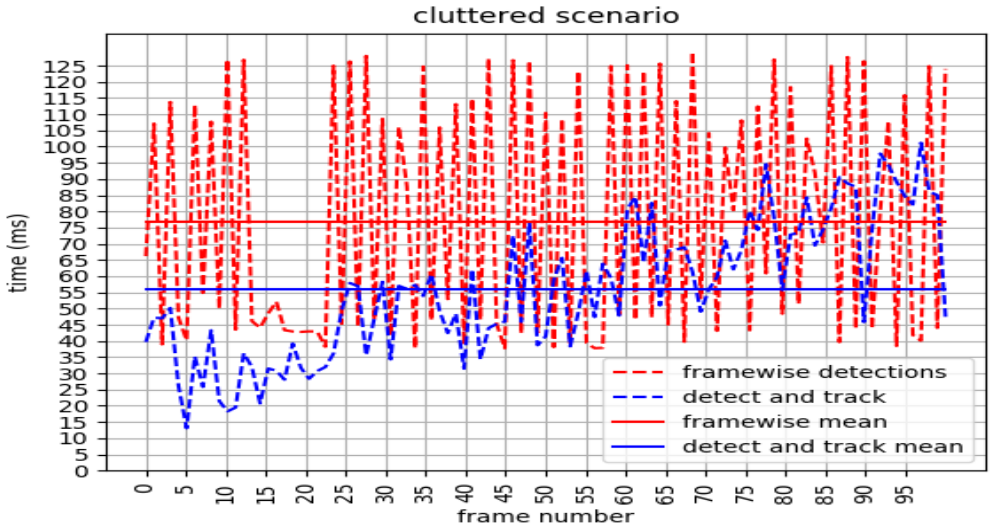


(a) SCS

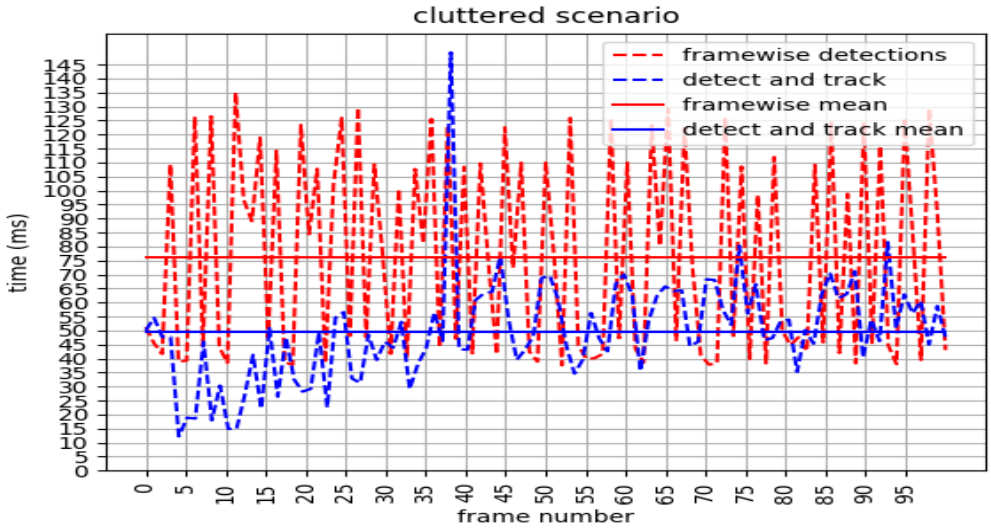


(b) SMS

Figure B.2: SCS and SMS Inference Time

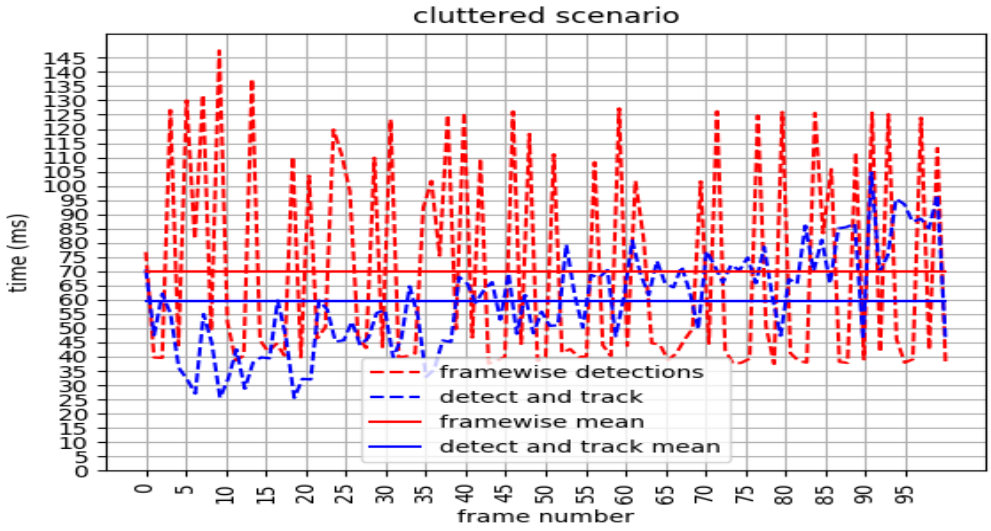


(a) CCC

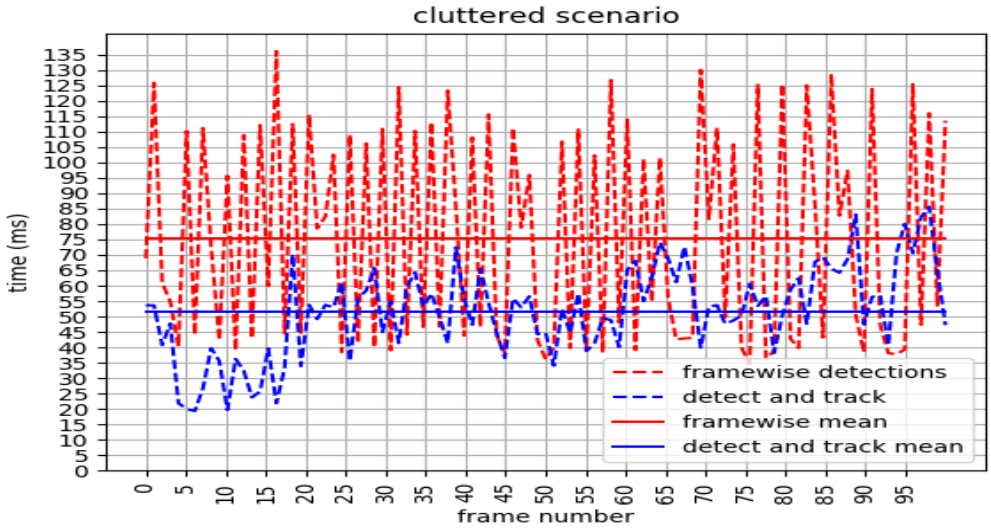


(b) CMC

Figure B.3: CCC and CMC Inference Time

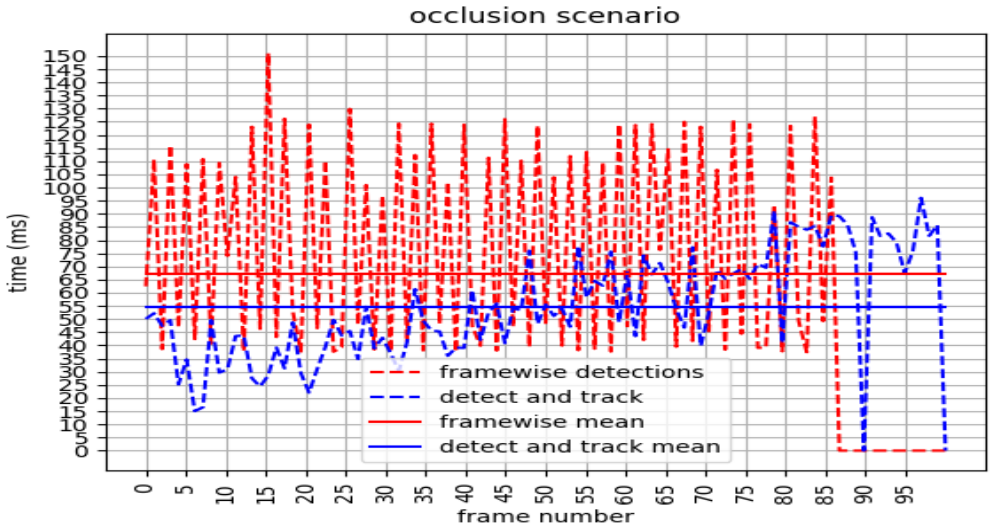


(a) SCC

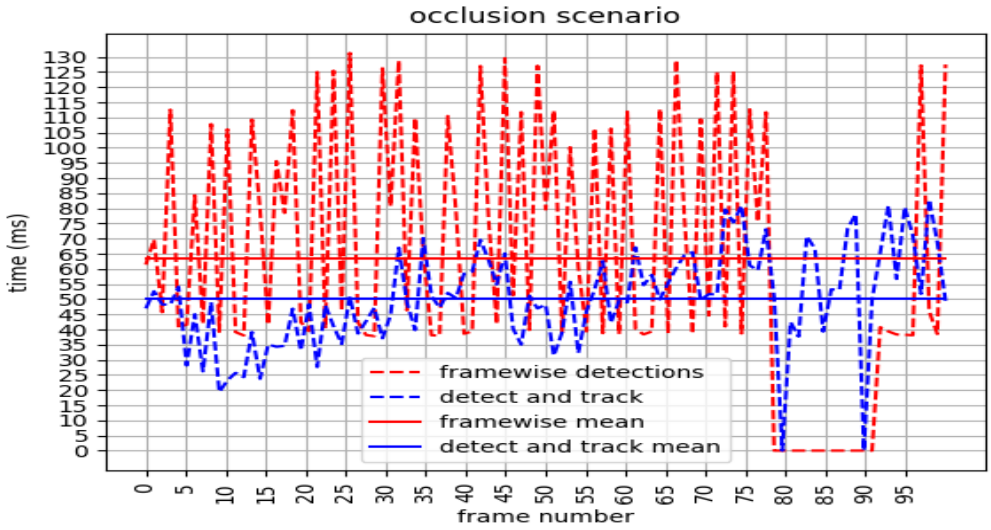


(b) SMC

Figure B.4: SCC and SMC Inference Time

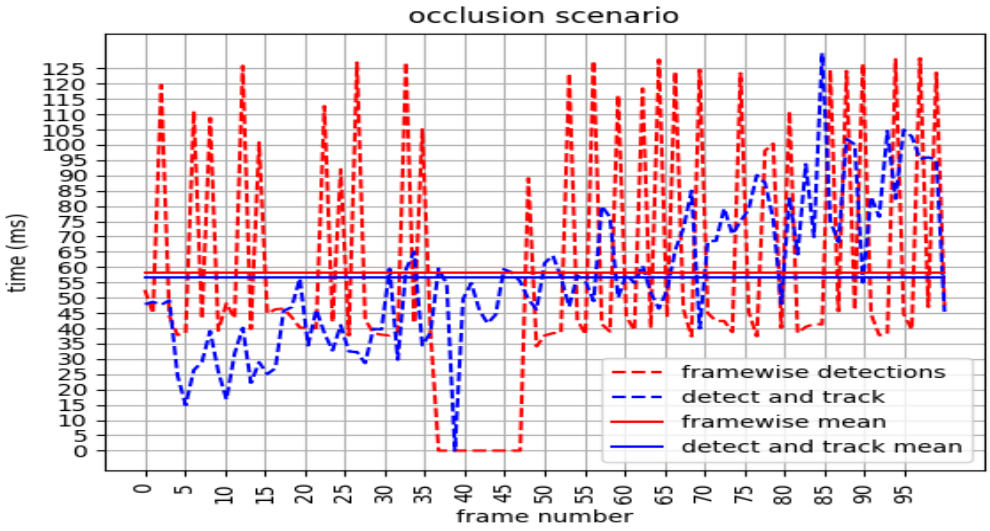


(a) CCO

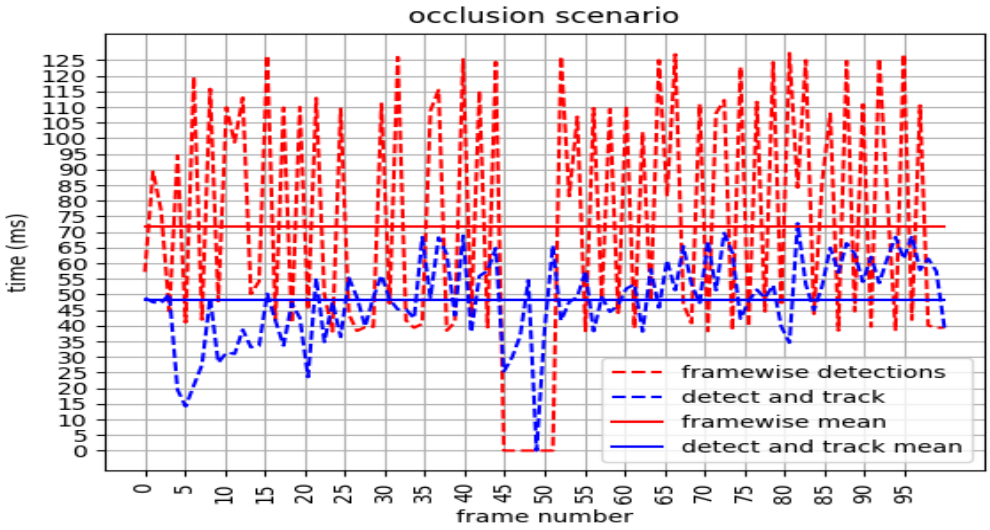


(b) CMO

Figure B.5: CCO and CMO Inference Time



(a) SCO



(b) SMO

Figure B.6: SCO and SMO Inference Time

C

Description of the Scripts

This project makes use of certain scripts to aid experimentation. This section would describe the functioning and details of those scripts.

C.1 Customised manual annotation

This script is used to annotate the frames of the input video and obtain the coordinates of the Ground Truth. It makes use of the OpenCV library[1]. The script reads the video as input and breaks the video into individual frames. These individual frames are then displayed sequentially to the user (each frame is displayed for 7 seconds) where they can draw the bounding box around the object and the scrip simply stores the coordinates of the top left corner and the bottom right corner. The duration of 7 seconds was reached upon by a series of trials and was found to be adequate amount of time to annotate the frame correctly. Figure C.1 depicts the individual frame of the video and Figure C.2 shows the frame with manual annotation.

As specified, the script stores the coordinates of the bounding box in a separate file titled *ground_truth.txt* by storing the coordinates for "Mouse Press" and "Mouse Release" events. Hence when user draws a bounding box around the object of interest, the coordinates for top left corner and bottom right corner are recorded following the OpenCV convention (origin is placed at top left corner of the window). One drawback of this method is that the annotator should always follow the same method for annotation i.e. start from the top left corner and finish the box at bottom right corner. If this convention is not followed the subsequent process might yield erroneous results. Owing to time constraints, a total of 50 frames per video were annotated using this script.

C.2 Calculation of Intersection over Union

This script is used to calculate the Intersection over Union of the approaches with the Ground

[1]Understanding inspired from: https://www.pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/

Figure C.1:
Frame without Ground Truth

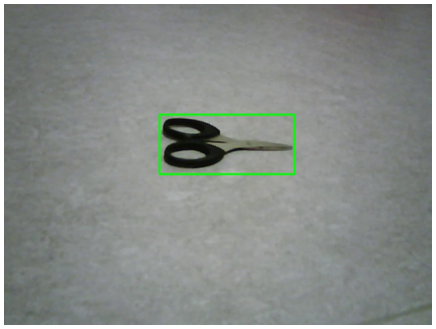


Figure C.2:
Frame with Ground Truth

Truth obtained from the previous script. This script[2] takes input as the values stored in the *ground_truth.txt* file of both the *Coupled* and *Framewise* approach. Using python functions intersection() and union(), the script implement the formula for calculation of IoU and generates a list of IoU values for 50 frames. This script also plots the values of IoU values over the course of these frames. The average over the 50 frames is also plotted on the same graph. The graphs generated from this script can be seen in **Appendix A**.

[2]https://stackoverflow.com/questions/25349178/calculating-percentage-of-bounding-box-overlap-for-image-detector-evaluation

References

[1] Bounding box predictions - deeplearning.ai — coursera. https://www.coursera.org/learn/convolutional-neural-networks/lecture/9EcTO/bounding-box-predictions.

[2] Kalman filter in one dimension. https://www.kalmanfilter.net/kalman1d.html, .

[3] Tracking — computer vision with python 1.0 documentation. https://www.hdm-stuttgart.de/~maucher/Python/ComputerVision/html/Tracking.html#kalman-filter, .

[4] How a kalman filter works, in pictures — bzarg. https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/, . (Accessed on 01/12/2020).

[5] aeb6bd253f65a32e6971f3d762f076b55f52.pdf. https://pdfs.semanticscholar.org/5051/aeb6bd253f65a32e6971f3d762f076b55f52.pdf.

[6] Mathwork website. https://de.mathworks.com/solutions/deep-learning/convolutional-neural-network.html.

[7] Understanding lstm networks – colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] Shivang Agarwal, Jean Ogier Du Terrail, and Frédéric Jurie. Recent advances in object detection in the age of deep convolutional neural networks. *arXiv preprint arXiv:1809.03193*, 2018.

[9] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[10] Alex Broad, Michael Jones, and Teng-Yok Lee. Recurrent multi-frame single shot detector for video object detection. In *BMVC*, page 94, 2018.

[11] A. Burton and J. Radford. *Thinking in Perspective: Critical Essays in the Study of Thought Processes*. Psychology in progress. Methuen, 1978. ISBN 9780416858402. URL https://books.google.de/books?id=CSgOAAAAQAAJ.

[12] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[14] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88 (2):303–338, 2010.

[15] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[16] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3038–3046, 2017.

[17] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick Van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[18] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465*, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Congrui Hetang, Hongwei Qin, Shaohui Liu, and Junjie Yan. Impression network for video object detection. *arXiv preprint arXiv:1712.05896*, 2017.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[26] Jeffrey Humpherys, Preston Redd, and Jeremy West. A fresh look at the kalman filter. *SIAM review*, 54(4):801–823, 2012.

[27] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.

[28] Rudolph Emil Kalman and Others. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[29] Kai Kang, Hongsheng Li, Tong Xiao, Wanli Ouyang, Junjie Yan, Xihui Liu, and Xiaogang Wang. Object detection in videos with tubelet proposal networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 727–735, 2017.

[30] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, et al. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907, 2017.

[31] Athindran Ramesh Kumar, Balaraman Ravindran, and Anand Raghunathan. Pack and detect: Fast object detection in videos using region-of-interest packing. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 150–156. ACM, 2019.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[33] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5686–5695, 2018.

[34] Mason Liu, Menglong Zhu, Marie White, Yinxiao Li, and Dmitry Kalenichenko. Looking fast and slow: Memory-guided mobile video object detection. *arXiv preprint arXiv:1903.10172*, 2019.

[35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[36] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. *arXiv preprint arXiv:1705.03550*, 2017.

[37] Yongyi Lu, Cewu Lu, and Chi-Keung Tang. Online video object detection using association lstm. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2344–2352, 2017.

[38] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

[39] Wartini Ng, Budiman Minasny, Maryam Montazerolghaem, José Padarian, Richard Ferguson, Scarlett Bailey, and Alex Mcbratney. Convolutional neural network for simultaneous prediction of several soil properties using visible/near-infrared, mid-infrared, and their combined spectra. *Geoderma*, 07 2019.

[40] Guanghan Ning, Zhi Zhang, Chen Huang, Xiaobo Ren, Haohong Wang, Canhui Cai, and Zhihai He. Spatially supervised recurrent convolutional neural networks for visual object tracking. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[41] Wanli Ouyang, Xiaogang Wang, Xingyu Zeng, Shi Qiu, Ping Luo, Yonglong Tian, Hongsheng Li, Shuo Yang, Zhe Wang, Chen-Change Loy, et al. Deepid-net: Deformable deep convolutional neural networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2015.

[42] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1536–1545, 2018.

[43] Dhara Patel and Saurabh Upadhyay. Optical flow measurement using lucas kanade method. *International Journal of Computer Applications*, 61(10):6–10, 2013.

[44] Esteban Real, Jonathon Shlens, Stefano Mazzocchi, Xin Pan, and Vincent Vanhoucke. Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5296–5305, 2017.

[45] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[46] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[48] Jiwon Ryu, Youngjin Moon, Jaesoon Choi, and Hee Chan Kim. A kalman-filter-based common algorithm approach for object detection in surgery scene to assist surgeon's situation awareness in robot-assisted laparoscopic surgery. *Journal of healthcare engineering*, 2018, 2018.

[49] Jing Shi and Chenliang Xu. Differential network for video object detection.

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[51] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 3119–3127, 2015.

[52] Fanyi Xiao and Yong Jae Lee. Video object detection with an aligned spatial-temporal memory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 485–501, 2018.

[53] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Craft objects from images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6043–6051, 2016.

[54] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 408–417, 2017.

[55] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2017.

[56] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards high performance video object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7210–7218, 2018.