SINGLE-SHOT OBJECT DETECTION AND CLASSIFICATION USING HAAR-LIKE FEATURE BASED RANDOM DECISION FOREST

A Thesis

by

Nekruzjon Maxudov

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
May 2017

SINGLE-SHOT OBJECT DETECTION AND CLASSIFICATION USING HAAR-LIKE FEATURE BASED RANDOM DECISION FOREST

Approved by:

Assistant Professor M. Furkan Kirac,
Advisor
Department of Computer Science
Özyeğin University

Professor A. Tanju Erdem
Department of Computer Science
Özyeğin University

Professor Lale Akarun
Department of Computer Engineering
Boğaziçi University

Date Approved: 31 May 2017

To my parents and my beloved wife Aysoltan

ABSTRACT

Object detection and tracking have been studied for decades and many algorithms have been introduced. Vision-based object detection and tracking became an important task with the increasing number of surveillance cameras. However, false alarm rates are still an issue to be solved in human operator managed scenarios. As precision and accuracy increase, false alarm rates become more manageable. In this thesis, a novel system for single-shot detection and classification of the object in images is introduced. For this purpose, we implemented Random Decision Forests (RDF) using Haar-like features. RDF and Haar-like feature calculation implemented on GPU are known for their test time speed. Thus, we are using RDFs for pixel level object classification, a methodology known for its balanced test-time performance both for speed and quality. The increase in accuracy is shown by conducting experiments on MNIST, INRIA, and PETS09 datasets. As a demonstrative application, we used proposed RDF for on-road vehicles detection and tracking. A Sequential Monte Carlo method based algorithm, also known as Particle Filter (PF), is implemented for tracking detected objects. For non-linear and non-Gaussian processes, PF is a powerful methodology and is easy and preferable to be implemented on GPU with RDF. The proposed system puts emphasis on real-time speed of the algorithm on conventional computers. Compared to YOLO (You Only Look Once), our method shows comparable vehicle detection accuracy and computational speed in a conventional computer. Moreover, we are introducing a new framework where different tracking algorithms can be implemented and tested. It provides various modules for data extraction, data generation, training and testing algorithms with different parameters. Usage of the modules in the framework is also discussed.

ÖZETÇE

Son yirmi yılda gözetim sistemleri sayısının artması ile görüye dayalı nesne algılama ve takip algoritmalarına yönelik araştırmalar hız kazanmıştır. Bu araştırmalar dahilinde nesne algılama ve takip problemleri birçok yönü ile incelenmiştir, fakat otomatik alarm üretimi problemleri için başarım oranı daha yüksek izleme algoritmalarına halen ihtiyaç duyulmaktadır. Gözetim sistemlerinin yanlış alarm vermesi, insan operatörlere dayalı sistemlerde henüz tamamen çözülmemiş bir problemdir. Nesne tanıma başarımının artması, yanlış alarm verilen durumların azalmasına neden olmaktadır. Bu tez kapsamında resimde bulunan objeleri tek-seferde saptayan ve sınıflandıran bir yöntem geliştirilmiştir. Rastgele Karar Ormanlarının (RDF) test hızının oldukça yüksek olduğu bilinmektedir. Bu bağlamda, önerilen sistem piksel seviyesinde obje sınıflandırma işlemini RDF ile yapmaktadır. Kullanılan RDF'in performansını arttırmak amacı ile Haar benzeri öznitelikleri kullanan bir RDF geliştrilimiştir. Haar özniteliklerinin kullanan RDF'in performansı arttırdığı el yazısı ile yazılmış olan rakamlardan oluşan MNIST, insan resimleri içeren INRIA ve PETS09 veritabanları üzerinde yapılan testler ile gösterilmiştir. Örnek bir uygulama alanı olarak, önerilen RDF araç saptama ve takibi problemi üzerinde test edilmiştir. Saptanan objeleri takip etmek için Sıralı Monte Carlo, diğer adıyla Parçacık Filtresi (PF) uygulanmıştır. Saptama aşamasında her bir piksel için elde edilen olabilirlik değerleri ile parçacıkların ağırlıkları hesaplanmıştır. PF kullanılmasının temel nedeni, Gauss ve çizgisel olmayan işlemler için güçlü bir algoritma olmasının yanı sıra RDF gibi GPU'ya uygun bir yapıya sahip olmasıdır. Önerilen takip sistemi standard bilgisayarlarda gerçek zamanlı uygulamaya ağırlık vermektedir. YOLO ile kıyaslandığında, önerilen yöntem araç saptama başarımında yakın bir performans sunarken, standard

bilgisayarlarda daha yüksek bir hıza sahip olduğu gözlemlenmiştir. Önerilen sisteme ek olarak bu tez kapsamında, farklı algoritmaları uygulamak ve test etmek için bir framework geliştirilmiştir. Bu framework, veri özütleme, veri oluşturma, algoritma eğitimi ve algoritmaları değişik parametrelerle test etmek için tasarlanan modüllerden oluşmaktadır. Tez içeriğinde modüllerin kullanımı ve fonksyonları anlatılmıştır.

ACKNOWLEDGEMENTS

During the continuation of my graduate study, I have had the privilege and honor of working with Dr. Furkan Kıraç, my research advisor, Dr. Ali Özer Ercan and Prof. Tanju Erdem. I have learned so much from their technical knowledge and research philosophy. They supported me throughout my research by providing their valuable time and knowledge. I consider myself extremely lucky to have a chance to work and learn from such a technical pioneers and wonderful people.

Special thanks to Dr. Furkan Kirac for his invaluable encouragement and support. I thank him for patiently listening to me on technical subjects or any other life-related topics and sharing his wisdom.

I would like to thank my peers at the OzU Vision and Graphics Lab for their valuable friendship and support. Working with them was a privilege and most memorable for me at OzU.

Special thanks to my parents and my wife. I am so grateful for their encouragements and sacrifices during my educational life. They were very supportive at all levels of my study. They helped me to go through all challenges without giving up.

This work has been done at Vision and Graphics Laboratory at Ozyegin University and supported with funds from Scientific and Technological Research Council of Turkey (TUBITAK) through the "2215 - Graduate Scholarship Programme for International Students" scholarship program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Multiple object tracking (MOT) is a fundamental research area in the field of computer vision [6][7][8]. It has been studied extensively for decades and many algorithms have been developed [9][10]. Recent advances in detection and tracking of multiple objects have led to its application to diverse practical problems such as visual surveillance, augmented reality and bio-medical imaging [8]. As the number of surveilance cameras on buildings, roads are increasing, the subject of object detection and tracking became even more popular. Nonetheless, it is not an easy task and complexity increases because of imprecise and noisy detections, occlusions by the other objects or background, and dynamic interactions among objects [11]. Despite the fact that numerous methods have been introduced in the literature, object detection and tracking is still a prominent problem.

Current technological advancements in deep learning allow us to detect most objects precisely[4][5]. Consequently, many recent studies on MOT adopt tracking-by-detection approaches [12][13] where the key research topic is data association to link object detections in a sequence of frames. However, data association problem is a complicated task on its own. Although this method has its advantages in handling complex images, it is difficult to achieve real-time performance. Furthermore, deep learning algorithms need big data and they are much more expensive computationally.

Mobile technologies have also become popular as well as efficient. As a result, fast object tracking has become essential. Most of the algorithms proposed in the literature need high computation power, which might not be feasible for the mobile applications. We are proposing an algorithm that puts emphasis on real-time speed

of the algorithm on the conventional computers. For this purpose, we decided to use two well known algorithms for their fast test-time speed in the literature: Random Decision Forests (RDF) [14] and Haar-like features [15]. Proposed algorithm for object detection and classification utilizes RDF that proposes a weak hypothesis of the bounding box with posterior distribution over the classes for each pixel in the input image in one pass. Combining these weak hypothesis proved to be more stable and unbiased in contrast to stronger alternatives. Designed RDF exploits a Haar-like feature set responses around the sample. Moreover, it is known that RDF is extremely fast on GPU due to its binary nature which makes it feasible to real time applications. By clustering this bounding box hypothesis map via Mean Shift, object detections with their posterior class probability distributions are obtained. The Haar-like feature is well suited to the rigid object detection [16], since rectangular features are sensitive to edges, bars, vertical and horizontal details, and symmetric structures. It has been widely used for face [15], pedestrian [17], vehicle [18] and hand pose detection and tracking algorithms [19][20]. Introduced algorithm was tested on MNIST handwritten digits, INRIA person and PETS09 pedestrian datasets and hight detection precisions were obtained. To test introduced algorithm for object tracking we used it in combination with Particle Filters (PF) [21], also known as the Sequential Monte Carlo, for single-shot vehicle detection and tracking. PF is implemented for tracking detected objects by RDF. PF is widely used for multi-object detection [22], multi-object segmentation, multi-object tracking [23][24], real-time simultaneous localization and mapping (SLAM) [25] etc. tasks. For non-linear and non-Gaussian processes, PF is fast, powerful methodology which is easy and preferable to be implemented on GPU with RDF. Likelihoods obtained from single-shot detection by RDF are used to compute the weights for the particles that are used for tracking vehicles. The usage of PF and RDF is fast and therefore suitable for doing computations on a conventional computer. This enables us to track vehicles in a responsive fashion

by using resources available in our conventional computers. The state of the art regarding object detection and tracking is explained in detail in Chapter 2.

CHAPTER II

PREVIOUS WORK

2.1 Object Detection

Object detection in static images is one of the fundamental tasks in computer vision and has been studied intensely during past few decades [15][26][27][22]. It is a challenging task to accomplish because of its complexity in identifying where the object is located. Detection has to take into account the image transformations and needs to be invariant to those changes. The knowledge of the object characteristics must be determined and learned by the detector in order to detect an object in an image. Thus, the most important stage in the object detection is feature extraction stage. For this purpose, different algorithms utilize features like color information, texture, edge orientation etc. With the advancement in computational power, we are now able to process and train huge amounts of data by using various machine learning algorithms efficiently. In general, object detection systems start with robust feature (Haar [15], SIFT [26], HOG [28], Convolutional [29]) extraction, which is followed by training classifiers for object identification in the feature space [5]. There are many different approaches used for this task. However, we can put them into two main categories: traditional methods and deep learning based methods.

2.1.1 Traditional Methods

With the popularity of mobile technology and its applications, traditional methods became even more important. Generally, traditional methods are computationally cheaper than deep learning methods in almost every scenario. There are lots of different algorithms in the literature [1][2][30][31]. The general outline is as follows:

- Extract robust features
- Pre-process and transform features
- Use transformed features to train some machine learning algorithm
- Use trained expert network for detection

Until now many varieties of features have been introduced in the literature but most commonly used features among all are Haar-like [15], HOG [28] and SIFT [26] features. Many traditional algorithms use one of these features with some modifications and preprocessing. Traditional methods generally use sliding windows for detection.

1. **Haar-like:** It is a feature similar to Haar wavelet introduced for object detection by Viola and Jones initially for face detection [15]. It is a weak classifier with one of the fastest implementation in the literature. A large number of Haar-like features are necessary to describe an object with sufficient accuracy. A Haar-like feature focuses on adjacent rectangular regions at a specific location in a detection window. It computes the sum of pixel intensities in each region and calculates the difference between these sums and is used to categorize subsections of an image. In Figure 1, example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles is subtracted from the sums of pixels in the gray rectangles.

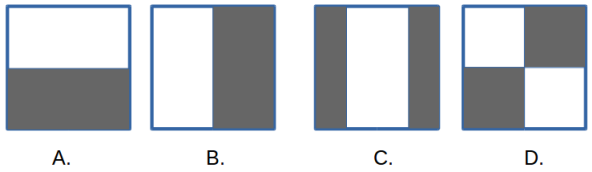
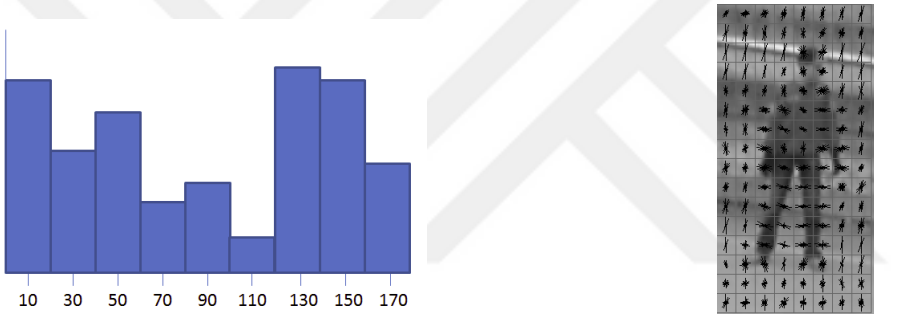


Figure 1: (a,b) Two-rectangle features (c) Three-rectangle feature (d) Four-rectangle feature

2. **HOG:** Histogram of oriented gradients is a type of feature descriptor introduced by Dalal and Triggs in 2005 for human detection [28]. The algorithm computes a gradient vector at each pixel. It slides an overlapping 16×16 window which is composed of four 8×8 cells. So, per cell, there are 64 gradient values which are quantized into 9-bin histograms. The histogram bins range from 0 to 180 degrees with step size 20 degrees per bin. Thus, for a 64×128 detection window, there are 7×15 blocks, 4 cells per block, and 9 bins per histogram that produce 3 780 value feature. The produced histogram is normalized to make it illumination invariant. This is widely established in computer vision literature.



(a) cell histogram (b) HOG visualized

Figure 2: HOG Features

3. **SIFT:** Scale-invariant feature transform is a type of feature descriptor introduced by David Lowe in 1999 [26]. In the image domain, it is invariant to translations, rotations, and scaling transformations. Moreover, it is robust to moderate perspective transformations and illumination variations. This descriptor is a position dependent histogram of local gradient directions around the interest point. Neighborhood histogram of gradients is computed and normalized to make it scale invariant. To make it rotation invariant, dominant vector is computed in the obtained neighborhood and the grid is reoriented based on computed dominant vector. Lowe stated that 4×4 grid is often a good choice based on experiments he conducted [26]. Therefore, local histograms computed

at each 4×4 grid cell pixels and 8 quantized directions lead to an image descriptor with $4 \times 4 \times 8 = 128$ dimensions per interest point.

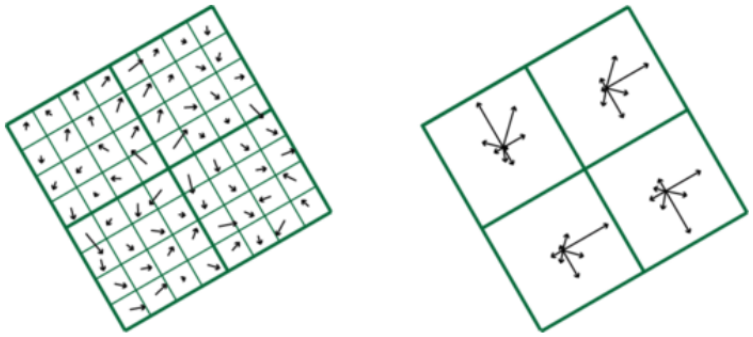


Figure 3: SIFT feature computed for 2×2 grid

Two most popular advanced examples of traditional detection algorithms are Deformable Part Model (DPM) [1] and Selective Search [2]. They both have been used for object detection and have demonstrated similar performances. They are still widely-used algorithms [3][32][33][34].

DPM is an algorithm between generative and discriminative model. It represents objects using mixtures of deformable part models. The first step in this algorithm is to construct a pyramid for different scale of the input image. It uses HOG features on pyramid levels before filtering [1]. Then, different root filters and part filters are used to get responses. Finally, classifiers are trained using latent SVM by combining responses and cost functions. Figure 4 shows a detection obtained with a single component person model is shown. It can be summarized as follows:

1. Strong low-level features based on histograms of oriented gradients (HOG)

2. Efficient matching algorithms for deformable part-based models (pictorial structures)

3. Discriminative learning with latent variables (latent SVM)

(a) (b) (c)

Figure 4: (a) coarse root filter (explanation needed) (b) several higher resolution part filters (c) spatial model for the location of each part relative to the root [1]

In contrast to DPM, Selective Search algorithm uses bag-of-words for object recognition [2], which are more complicated and stronger features. To compute feature, it samples descriptors at each pixel on a single scale. Used codebook size is four thouthand and pyramid with 4 levels 1×1, 2×2, 3×3 and 4×4 division. Produced feature vector length is three hundtred sixty thouthand. This descriptor is more representative of deformable object types due to coarser spatial subdivisions. Finally, classifiers are trained using Support Vector Machines (SVM) [35] with a histogram intersection kernel. The training procedure is illustrated in Figure 5.

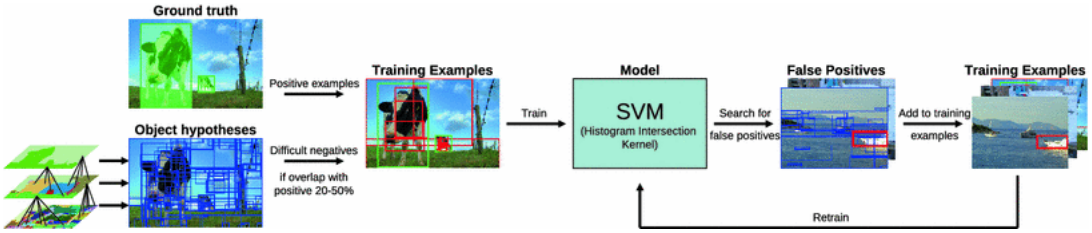


Figure 5: Selective Search training procedure [2]

2.1.2 Deep Learning Based Methods

Despite its long history, deep learning was not that popular and widely used until last decade, because of the computational cost of the artificial neural networks (ANN) and lack of big enough training data. Moreover, it was not yet completely understood

that a patient training using weak features of greater numbers would beat a relatively smaller number of more complex features. Advances in hardware and processing power brought by powerful graphics processing units (GPUs) played an important role in the resurgence of deep learning. The other important turning point in using deep learning in object detection was brought by the availability of huge datasets presented by big corporations such as Google [36], Facebook [37], Microsoft [38], etc.. Deep learning (Convolutional Neural Networks) techniques became even more popular after the introduction of Region-CNN by Ross Girshick [39] and became a state of the art methodology in object detection in single-shot image detection. Today in terms of accuracy they dominate object detection benchmarks like PASCAL VOC [40] and Microsoft COCO [38]. State of the art algorithms in object detection and recognition are: Faster R-CNN [3], SSD [4] and YOLO [5].

Faster R-CNN is the extension of the R-CNN introduced by Ross Girshick [39]. It follows the steps as shown in the figure below.

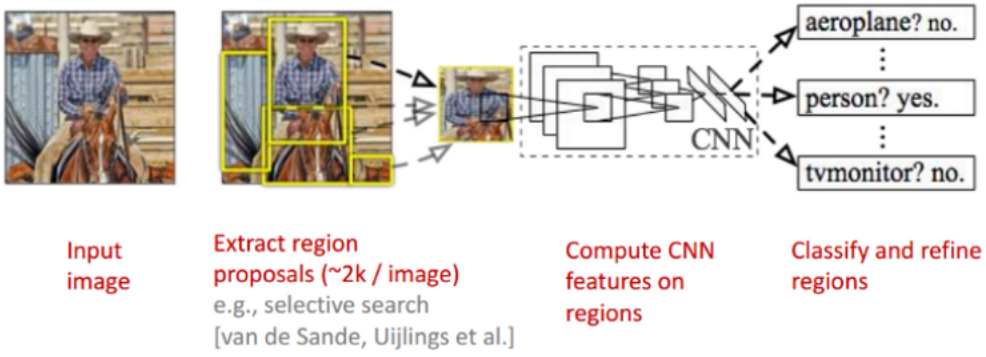


Figure 6: Region-based Convolution Networks (R-CNNs) [3]

It is composed of two modules [3]:

1. Region Proposal Networks (*RPNs*) module
2. Fast R-CNN module

The RPN module is a fully convolutional network that processes regions and tells the Fast R-CNN module where to look. The Fast R-CNN processes proposed regions and detect objects. They are unified as a single object detector network as shown below.

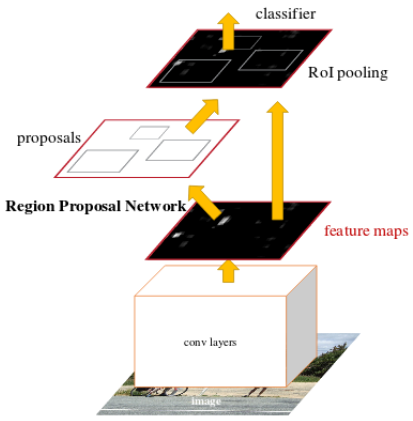
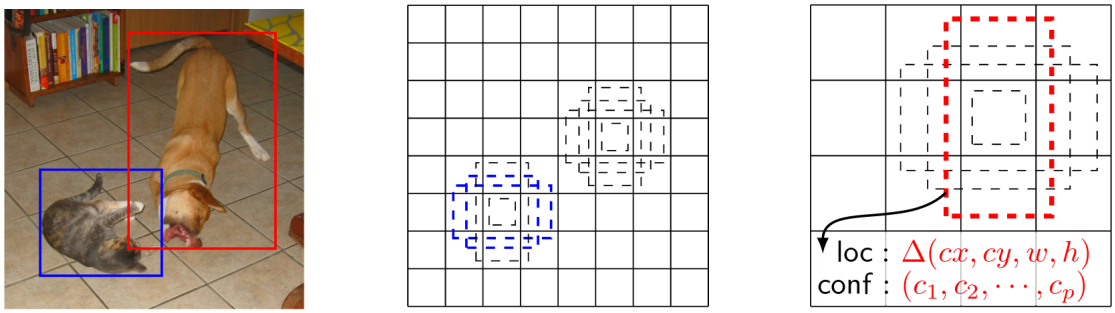


Figure 7: Faster R-CNN Detection System [3]

Another algorithm is Single Shot MultiBox Detector (SSD) [4] which is the other popular CNN algorithm. The idea behind this algorithm is similar to the usage of RPN module [3] in Faster R-CNN algorithm. For detection predictions, it uses fixed sized boxes similar to anchor boxes used in Faster R-CNN [3]; however, it is applied to several feature maps of different resolutions. The space of possible output box shapes is efficiently discretized by allowing different default box shapes. Since scores for each category are computed simultaneously, the need for extra classifier for combining region predictor and core CNN is avoided. Therefore, training is much faster, easier and provides easy integration with other applications [4].

Another deep learning algorithm that we stated is YOLO [5] which was introduced in CVPR 2016. Unlike other systems like R-CNN [5], which require thousands of networks for a single image, YOLO makes predictions with a single network evaluation pass. It uses information that has a global context in the image during prediction



(a) Image with GT boxes (b) $8x8$ feature map (c) $4x4$ feature map

Figure 8: SSD framework [4]

stage. Prediction steps are done by applying a single neural network to the full image. The applied network divides the image into regions and predicts bounding boxes and probabilities for each region. Then, these bounding boxes are weighted by the predicted probabilities.

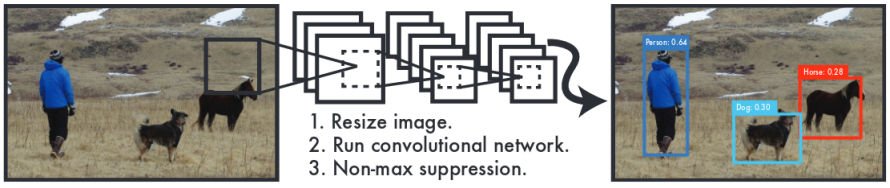


Figure 9: YOLO Detection System [5]

Detection system works as follows:

1. resizes the input image to 448×448
2. runs a single convolutional network on the image
3. thresholds the resulting detections by the model's confidence

The YOLO models detection as a regression problem [5]. It divides the image into a $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B*5+C)$ tensor [5].

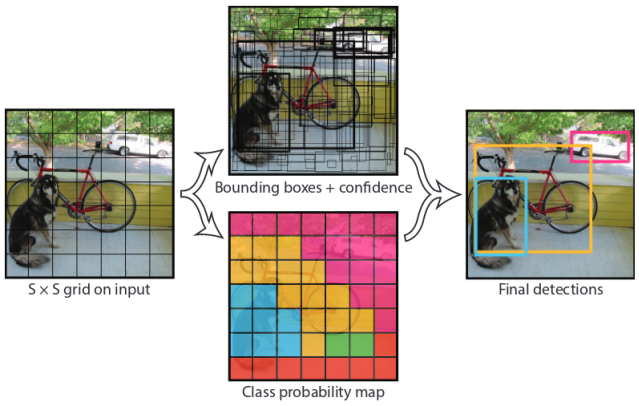


Figure 10: YOLO Detection Model [5]

2.2 Tracking

After the detection has been computed, we need to track objects. Over the past decade, a large amount of work that has been devoted to multi-modal object tracking algorithms [9]. If any object detector introduced in proceeding section is used, the problem of tracking becomes data association of the detection results. This approach has become the recent popular method [12][13][41]. However, data association problem is a complicated task on its own [8]. Numerous algorithms have been developed to reduce complexity and provide better algorithmic speed [41][42][43]. Although this method has its advantages on handling complex images, alleviating drift and processing temporary disappearance of objects, it is difficult to achieve real-time performance.

Nevertheless, most tracking algorithms in the literature are focused on only one object class, e.g. tracking either cars or people in the scene. Therefore, applying such heavy algorithms might be considered overkill for these kinds of problems. The current problem has generally been approached by applying some kind of background subtracter to identify foreground images, which most probably are the objects we are interested in. The, connected component analysis is used to obtain individual moving

blobs which are tracked using Kalman Filter (KF)[8]. The usage of KF is one of the earliest techniques introduced in the literature, which is a recursive method that predicts the current object position based on information from the previous frame. It is considered as one of the optimal methods assuming everything operating under Gaussian probability distribution; however, real world cases are usually non-Gaussian [23].

On the one hand, with non-Gaussian problems, it is almost impossible to evaluate the distribution analytically. On the other hand, particle filters (PF), also known as the Sequential Monte Carlo [44], recursively construct the posterior PDF of the state space using the Monte Carlo integration. Particle filter samples from proposal distribution in order to get a group of weighted particles for representing current status. Methods using PF are fast [23]. The improvement in object detection algorithms made PF popular in multi-object tracking algorithms. For example, Okuma et al. [21] introduced the Haar feature-based cascade classifier to particle filtering framework. They used the classifier as a detector to discriminate the object and background. Michael et al. [45] proposed a multi-object tracking algorithm based on PF and detection score, introducing detection score into the calculation formula of the matching score in data association. Gall et al. [22] introduced Hough Forests by using confidence as observation for particle filter.

CHAPTER III

PROPOSED METHOD

3.1 Overview

Random Decision Forest (RDF) [14] have been used extensively in the literature. In computer vision they have been used for tasks like hand pose estimation [46] and pedestrian detection. Methods similar to the algorithm proposed in this thesis are Hough Forests [22], RDFs that train local experts (SVMs) in each node [47], and RDFs that are trained as joint classification-regression model [48]. These methods use high level features for training which makes them computationally expansive. In this thesis we introduce RDFs that uses Haar-like features, which are very fast to compute. Due to the binary nature of RDF and fast computation time of Haar-like features, proposed method is fast and easily deployable to GPU for even faster computation speed. To the best of our knowledge, Haar-like feature has not been used in combination with RDFs in the literature. Moreover, the application of proposed method on vehicle tracking is also introduced. The application utilizes proposed Haar-RDF and Particle Filter (PF) [44] for detecting and tracking vehicles (Figure 11). Each particle has observation weight that has to be set. The accuracy of tracking heavily depends on these observations. Haar-RDF produces likelihood of the region containing tracked object. These likelihoods are used as observation values for particles. In the following subsection we, are going to elaborate on Haar-RDF, PF, and on how we perform training and updating particles in detail. In addition to that, algorithm verification methods are also explained. In the end, proposed framework for data extraction, training, detection and tracking is introduced. We are going to thoroughly explain each module in the framework.

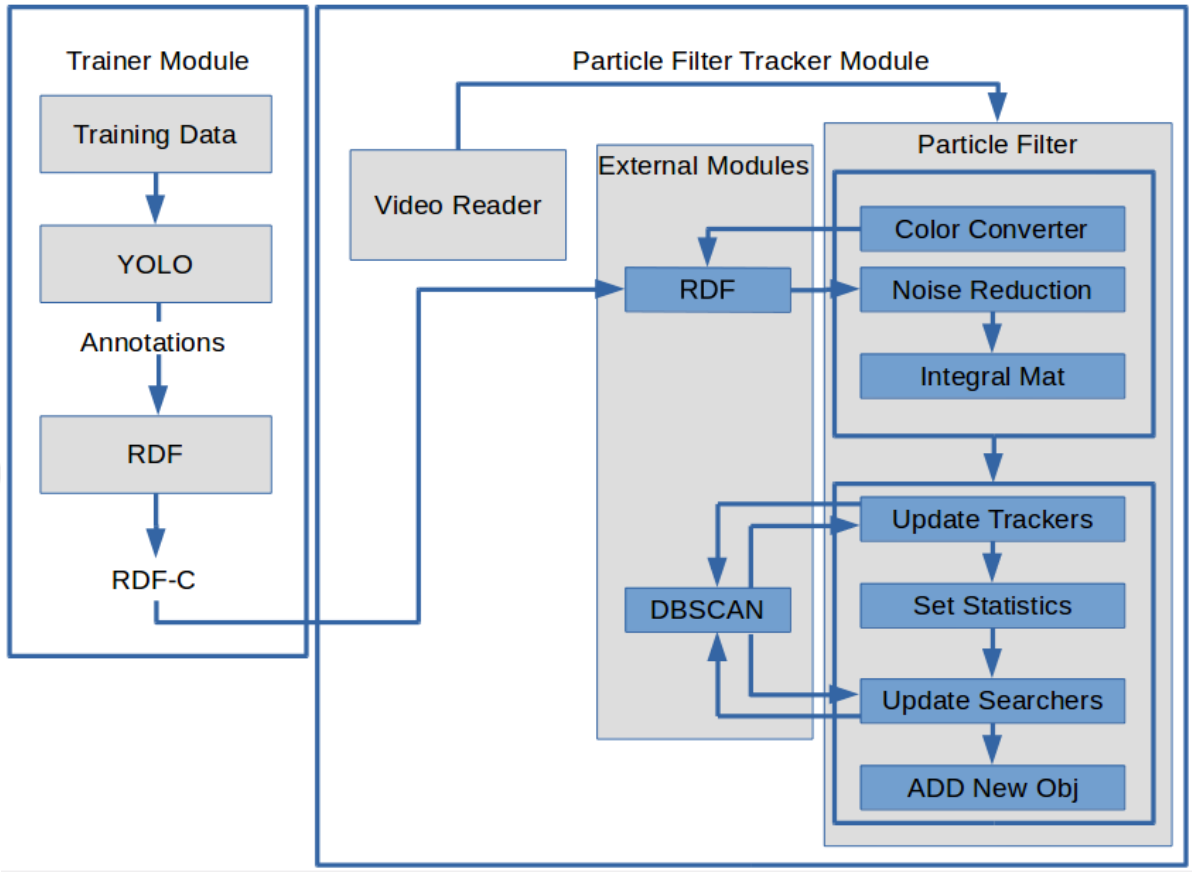


Figure 11: General Structure of the Proposed System

3.2 Random Decision Forest

Random Decision Forest (RDF) is an ensemble learning method which has been studied extensively in the literature [14][49]. They are extremely fast classifiers due to their binary structure. Tim Kam Ho was first to develop an RDF algorithm using random subspace method [49]. The RDF model proposed in this thesis uses the distance between pixel intensities and Haar [15] features to classify objects. These features are weak classifiers but when combined, similar to the AdaBoost [50] algorithm, they become strong. The implementation of the proposed RDF methodology is inspired by the RDF structure used in [46]. In this approach, each pixel will vote for object class label with a likelihood and vote with a higher average likelihood is used to determine the object label. RDFs are a method that uses multiple deep decision trees, trained

on randomly sampled parts of the same training set, to reduce the variance of the model [51].

Each tree in the forest is composed of two parts: decision nodes and leaf nodes [46]. Decision nodes are used to analyze the data and propagate the incoming input to one of its children according to the split criterion. Leaf nodes are nodes in the last level of the tree and based on the statistics collected from training data, they are used to infer the posterior probability of the class label. Splitting in a decision node is performed based on the following function:

$$f_n(F_n) < T_n \tag{1}$$

where F_n is input features, $f_n(F_n)$ is a function that produces a comparable value from input features and T_n is a threshold, combined together they produce a split criterion function (Eq. 1). Parameters for the decision node are learned after performing the non-improving epoch iteration procedure. $f_n(F_n) = T_n$ is a hyperplane in the feature space. The test decides which side of hyperplane the feature belongs to. Therefore, training involves determining test parameters and collecting statistics from a training set in a supervised manner [46]. Proposed RDF training algorithm is described in Algorithm 1.

3.2.1 Random Decision Trees

Random Decision Trees are highly non-linear predictors that hierarchically partitions feature space in to subsets S_n to obtain pure partitions at leaf nodes. Where, S_n represents the subset at n^{th} node. One of the most common ways to represent pureness mathematically is the Shannon entropy(Eq. 2). Where, $p(c)$ is the probability of class c in the subset S. Minimum entropy means that the partition is pure indicating that only one type of sample exists in that partition.

Algorithm 1 RDF Training

```
1: procedure TRAIN_TREES
2: for all tree in forest do
3:     set data and parameters
4:     procedure TRAIN
5:         initializes parameters
6:         initializes nodes
7:         Generates sub samples from each frame so that positive
8:         and negative pixel counts are equal
9:         procedure CONSTRUCT_TREE
10:            construct root node
11:            construct decision nodes
12:            compute leaf node histograms
13:        procedure COMPRESS_TREE
14:            travers tree preorderly starting at root
15:            set node values to compressed tree
16: procedure SAVE FOREST
```

$$H(S) = -\sum_{c \in C} p(c) log(p(c)) \tag{2}$$

In the binary case, partitioning is done via binary boundaries in the feature space as defined in Eq. 3. Where, $f_n(x)$ refers to a feature function, τ_n is the split threshold , S_i^L and S_i^R are defined as subsets of S_i at left and right child respectively.

$$\Psi_n(\rho) = \begin{cases} S^L & if \quad f_n(x) < \tau_n \\ S^R & if \quad f_n(x) \geqslant \tau_n \end{cases} \tag{3}$$

During training, $f_n(x)$ and τ_n are selected in a way that the entropy is minimized at each partition. This is achieved by randomly searching $f_n(x)$ and τ_n such that maximum gain(Eq. 4) is obtained. Feature function $f_n(x)$ can be designed due to the problem at hand. Ensemble of independently trained RDT's are called a forest (RDF) where posterior distributions are accumulated during test time providing more robust data.

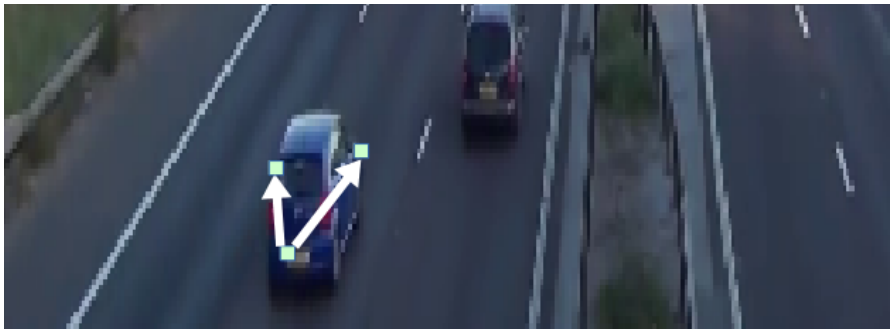


Figure 12: Sample pixel clasiffication on decision tree node

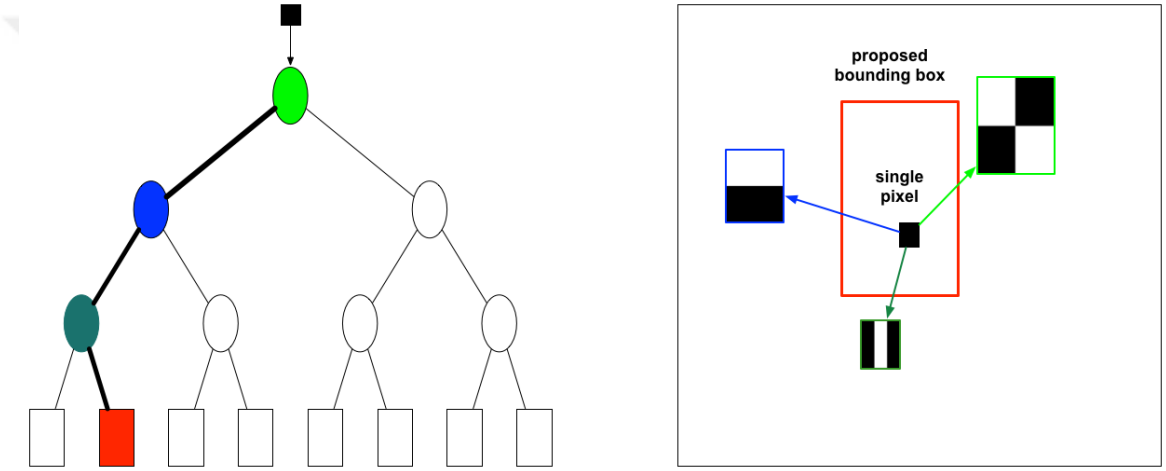


Figure 13: An example of complex Haar-RDF process during testing. Leaves are represented as square nodes in the tree. On left, each vector and haar feature is the $f_n(x)$ for each node with the same color.

$$I(S) = -\sum_{i \in \{L,R\}} \frac{|S^i|}{|S|} H(S^i) \tag{4}$$

Tree Node parameters are ID, τ, θ_{one}, θ_{two}, *Feature ID, isLeaf and leaf_histogram.* In each iteration, random θ and τ values are generated and using all available Haar-like features (Figure 14) the best one that partitions subset in a way that the information gain is maximized at the given node is selected. These iterations are continued until iteration count reaches its maximum value. If new values with better entropy division are detected, the iteration count is reset to zero. This is called a non-improving epoch iteration procedure. After the tree training is complete, something similar to (Figure

15) is generated. While testing, each pixel (Figure 12) goes through each node (Figure 13) and after reaching the leaf node, uses the obtained histogram to vote for the pixel label.

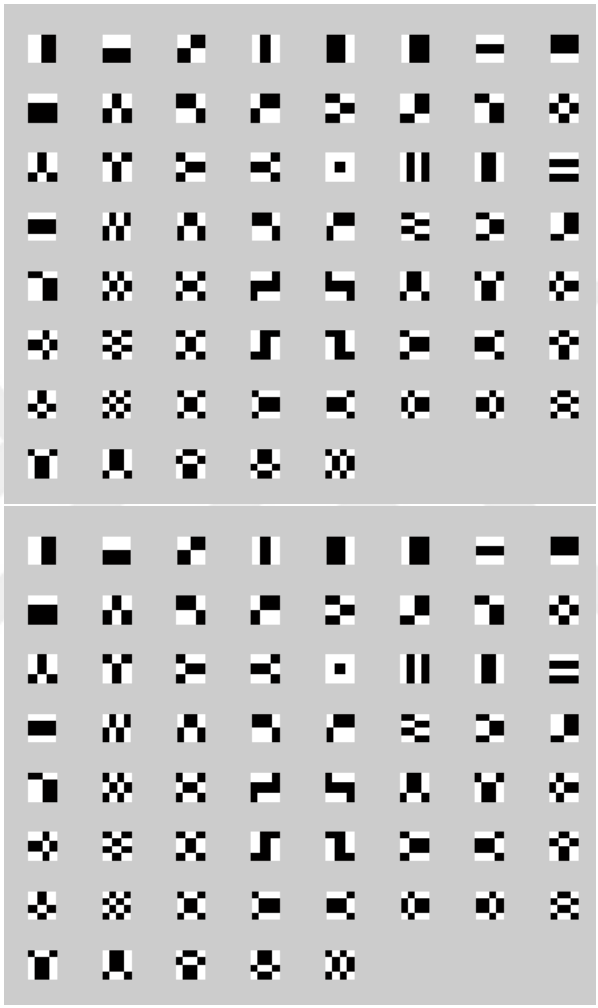


Figure 14: Haar-like Features

3.2.2 Compressed Random Decision Forest (Compressed-RDF)

After the tree is constructed, it is compressed in order to get rid of unnecessary training details. The compressed tree is a single matrix composed of K columns ($K = max(seven, label\_count)$) (Figure 16). The compression is done by traversing the tree using preorder traversal algorithm. It starts with the root node. In each

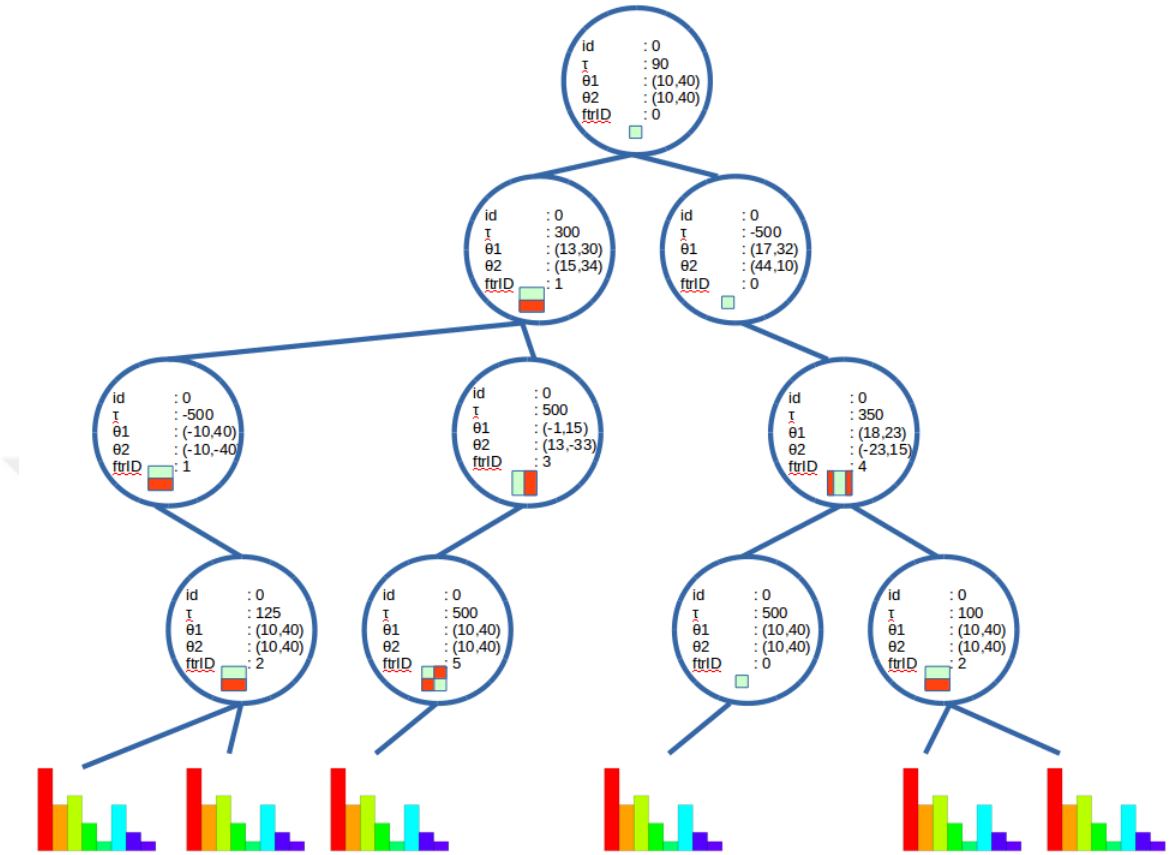


Figure 15: Sample tree and its leaves that stores object label likelihoods

given node we check whether the node is a decision node or a leaf node. If it is a leaf node, we put a sentinel value (-1) in the first column of the node and all other columns are used to store this leafs histogram. If it is the decision node we put left childs ID in the first column, feature ID in the second column, τ value in the third column, θ_{one} and θ_{one} values in the rest of the columns.

Consequently, the trained and compressed forest is ready for exploitation. The compressed forest is used which is much lighter than the regular RDF class, for detection purpose. Compressed-RDF contains compressed trees and uses CUDA kernel for the pixel classification if CUDA compatible GPU is present. The Compressed-RDF with GPU kernel is eight times faster that the Compressed-RDF with CPU classifier, in a conventional laptop GPU (GeForce GTX 860M/PCIe/SSE2).

Right Child ID	Feature ID	τ	θ1.x	θ2.y	θ2.x	θ2.y
30	0	90	10	40	-10	-40
5	2	150	-13	12	-34	-25
....						
-1	0.2	0.7	0.1	0	0	0

Figure 16: Sample compressed Tree

3.3 Particle Filter

Particle Filter (PF) [44], also known as the Sequential Monte Carlo (SMC), is well-known filtering algorithm used for solving Hidden Markov Chain (HMM) and non-linear filtering problems [23][24][44][52]. PF is used to estimate the internal state of the dynamic system based on partial observations. The idea is to calculate the posterior distributions of the states of some Markov process. Given some noisy observation, using a set of particles, it attempts at representing the posterior distribution of some stochastic process by applying a generic type mutation-selection sampling approach. The mutation-selection sampling algorithm is used for implementation of prediction-updating transition process [52].

The set of particles in PF represent samples from the distribution. Each particle has a likelihood weight assigned to it. This weight represents the probability of that particle being sampled from the probability density function. After undergoing re-sampling step, particles with higher weights replace particles with negligible weights [44].

Particle filtering is a powerful algorithm used for non-linear,multi-modal, and non-Gaussian processes. It is used in a variety of different fields. They are widely used in target tracking [23], economics [53][54], neuroscience [55] and biochemical networks

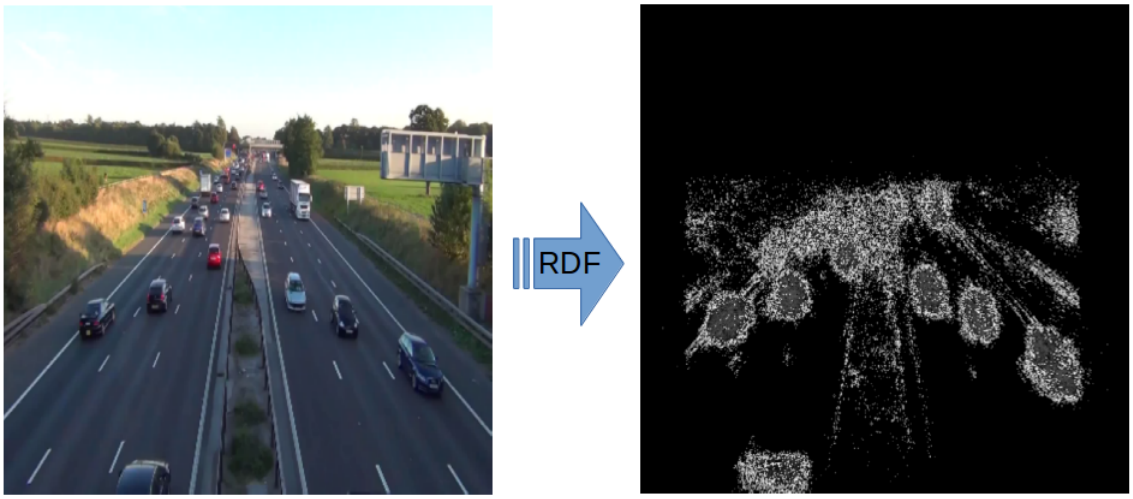


Figure 17: Random Decision Forest pixel clasification: (left) input image, (right) RDF pixel classification result

[56] to name a few.

The proposed implementation of PF in this thesis is similar to the original implementation explained in [44]. We have sample set S :

$$S = \{(s^n)|n = 1...N\} \tag{5}$$

and we are trying to approximate the probability distribution by this weighted sample set S, where s^n is n^{th} sample, and N is the number of samples. Each sample s represents one hypothetical state of the object, with a corresponding discrete sampling probability π, where

$$\pi^n = p(z_t|X_t = s_t^n) \tag{6}$$

$$\sum_{n=1}^{N} \pi^n = 1 \tag{7}$$

In our model, each particle has a rectangle shape with 2D position (x, y), width and height (w, h), velocity (dx, dy) and on video frame parameters. So, n^{th} sample

at time t would be:

$$s_t^n = \begin{bmatrix} x_t^n & y_t^n & dx_t^n & dy_t^n & w_t^n & h_t^n & \pi_t^n \end{bmatrix}^T \tag{8}$$

Let vector X_t be the state of tracked objects and vector Z_t all observations up to time t. Then, based on the observations of each particle, the probability π is calculated using Equation 6.

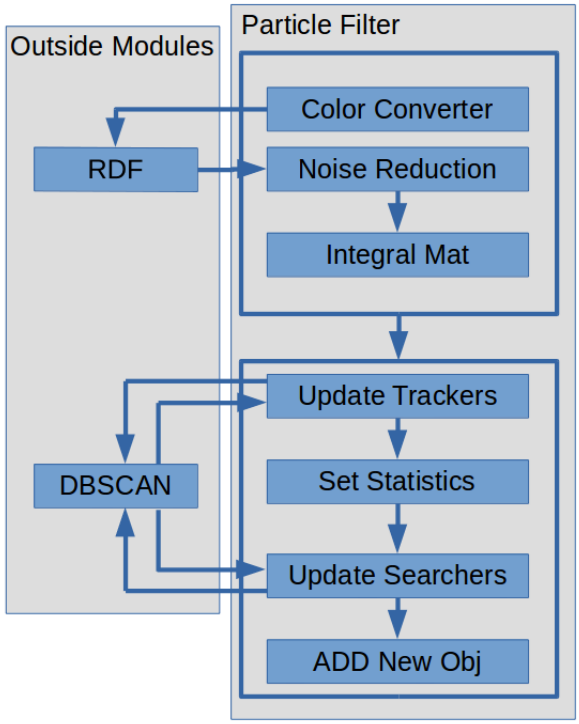


Figure 18: Particle-Filter block diagram

Proposed PF tracking model is composed of searcher particles and tracker particles. Searcher particles are used to detect the new vehicle in each step. They are randomly reinitialized when new frame is received. Tracker particles are used to track the detected object by searcher particles. When the new frame received, they are resampled M times based on their weights computed in the previous stage. The value of M depends on the number of clusters in the previous stage and a total number of trackers initialized per object ($M = nClusters * nTrackersPerObj$). This is an

important procedure which ensures that after objects disappear, trackers that are moved to other clusters after re-sampling are removed. If this step is skipped the number of trackers can grow to infinity and algorithm will stop functioning. The general steps in this particle filter tracker are as shown in the Algorithm 2.

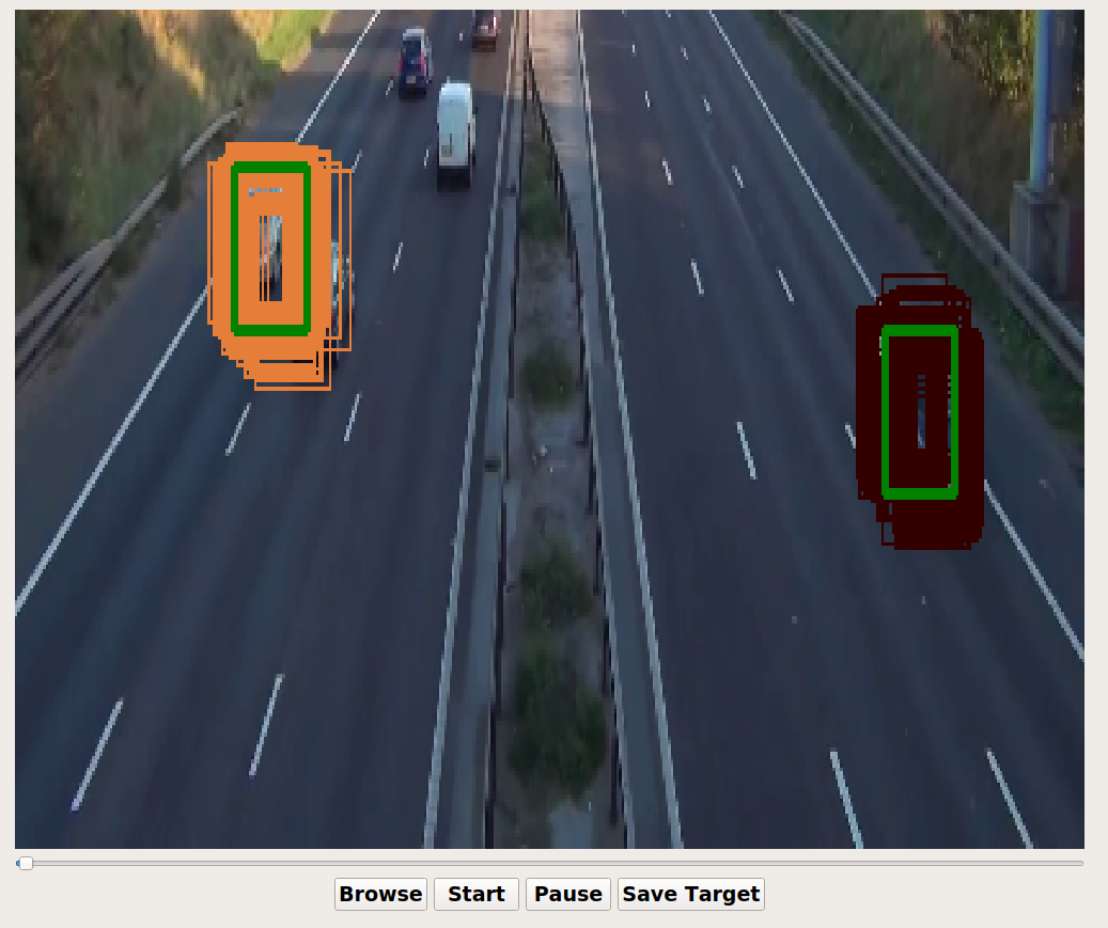


Figure 19: Sample output of searcher particles and their clustering

The model is initialized with an average width and height. The assumption for our model is that the size of tracked object change according to Gaussian distribution $N(\mu, \sigma)$ and velocity is constant. In each stage, we calculate object clusters by clustering our particles and collect statistics for the model update stage according to (Eq. 9) (Eq. 10).

Algorithm 2 Particle Filter Tracking

procedure EXEC() IS FIRED
    new frame is received
procedure COLOR SPACE CONVERSION
    if *frame color space* $\neq$ *RDF color space* **then**
        convert frame color space to RDF color space
procedure LIKELIHOOD MATRIX CALCULATION
    $likelihoodMat \leftarrow RDF$
procedure NOISE REDUCTION
    median filtering is performed on the matrix produced by RDF to reduce noise
procedure INTEGRAL IMAGE CALCULATION
    integral image is calculated for faster computation of future particle weights
procedure UPDATE TRACKER PARTICLES
    resample tracker particles
    compute weights
    tracked clusters $\leftarrow DBSCAN$
procedure CALCULATE NOISE STATISTICS
    based on cluster distribution, standard deviation of particle positions
    and standard deviation of width and heights are calculated and set for
    each particle
procedure OBJECT DETECTION (FIGURE 19)
    randomly inilialize N searcher particles
    for i **in** K **do**
        compute weights
        resample searcher particles
    compute weights
    sort tracker particles
    select top M particles $\rightarrow DBSCAN$
    detected clusters $\leftarrow DBSCAN$
procedure ADD NEW TRACKERS
    for *cluster* **:** *detected clusters* **do**
        if *tracked clusters* **contain** *cluster* **then continue**
        if $cluster.weight < minWeight$ **then continue**
        initialize L tracker particles around new detected object
procedure UPDATE TRACKER PARTICLES
    compute weights for tracker particles
procedure DETECTED REGIONS ARE DRAWN ON THE OUTPUT FRAME

$$s_{t+1} = s_t + U \tag{9}$$

$$\underbrace{\begin{bmatrix} x_t + dx_t \\ y_t + dy_t \\ dx + N(\mu_x, \sigma_x) \\ dy + N(\mu_y, \sigma_y) \\ w_t + N(\mu, \sigma) \\ h_t + N(\mu, \sigma) \\ \pi_t - \pi_t \end{bmatrix}}_{s_{t+1}} = \underbrace{\begin{bmatrix} x_t \\ y_t \\ dx \\ dy \\ w_t \\ h_t \\ \pi_t \end{bmatrix}}_{s_t} + \underbrace{\begin{bmatrix} dx_t \\ dy_t \\ N(\mu_x, \sigma_x) \\ N(\mu_y, \sigma_y) \\ N(\mu, \sigma) \\ N(\mu, \sigma) \\ -\pi_t \end{bmatrix}}_{U} \tag{10}$$

N is a normal distribution where (μ_x, μ_y) is the center of the particle cluster to which given particle belongs and (σ_x, σ_y) is a standard deviation of the particles belonging to the same cluster.

3.4 Framework

We have developed a framework with a friendly graphical user interface for recognition and tracking tasks. Here we are going to provide a detailed explanation on each module. It is composed of:

1. HOG Extractor Module: used to extract HOG features from input images
2. Data Extractor Module: used to extract training images from input video stream
3. RDF Trainer Module: used to train, test and save RDF for later usage in detection
4. PF Tracking Module: used to track car (object) using pre-trained detectors

3.4.1 Histogram of Gaussian Extractor Module

Histogram of Gaussian (HOG) [28] extractor module just asks for the dataset folder and extracts HOG features for the given folder. After extraction is complete, it saves data to the specified folder. To activate you press "Hog Feature Extraction" last button in (Figure 20)

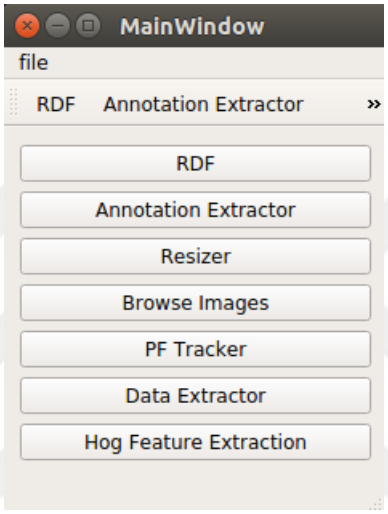


Figure 20: Histogram of Gaussian Extractor User Interface

This module is used for extracting HOG features to train a Support Vector Machine (SVM) [35]. This trained SVM was used for the validation of implemented particle filter algorithm. More detailed explanation is presented in Experiments section.

3.4.2 Data Extractor Module

This module is used to collect data for training both Random Decision Forests (RDFs). It uses the pre-trained SVN Classifier, the Background Subtracter, and the Blob Detector to detect all moving objects and save cropped detections Algorithm 3. The first thing we do for each frame is obtaining foreground pixels using Mixture of Gaussian (MOG2) [57], a Gaussian Mixture-based Background/Foreground Segmentation Algorithm which is robust to scenes due to illumination changes. Next, we

feed obtained binary foreground/background image and send it to a Blob Detector by providing minimum area and shape of the objects. Obtained blobs are put to test based on their size and prior trained SVM detection results. If a positive response is received, then we crop and save given blobs ROI. The position of the object in the scene is also registered for single frame detection RDF training.

Algorithm 3 Data Extraction

```
1: MOG2 ← Frame
2: BlobDetector ← MOG2
3: blobs ← BlobDetector
4: for all blob in blobs do
5:     if the blob is not valid then continue
6:     SVN ← blob
7:     if the result is positive then save detection
```

3.4.3 Random Decision Forest Trainer module

We are using Random Decision Forest (RDF) trainer module for training random decision forests and testing them. It facilitates parameter tuning to train optimal RDF. We are able to save trained RDFs. They can, then, be used for testing with provided input data. Our RDF Module is composed of two main sections: Train (Figure 22), Test (Figure 23).

These two sections share 4 subsections:

1. Data Loader. It is used to select dataset type, set the maximum allowed data, set skip count, set a color, set $L^*A^*B^*$ conversion and display ground truth data.

   - Since for our experiments we are using different datasets, we need to specify which data we want to load. Each dataset is stored differently and therefore should be handled accordingly while loading. This section supports this feature to facilitate user with a better experience.

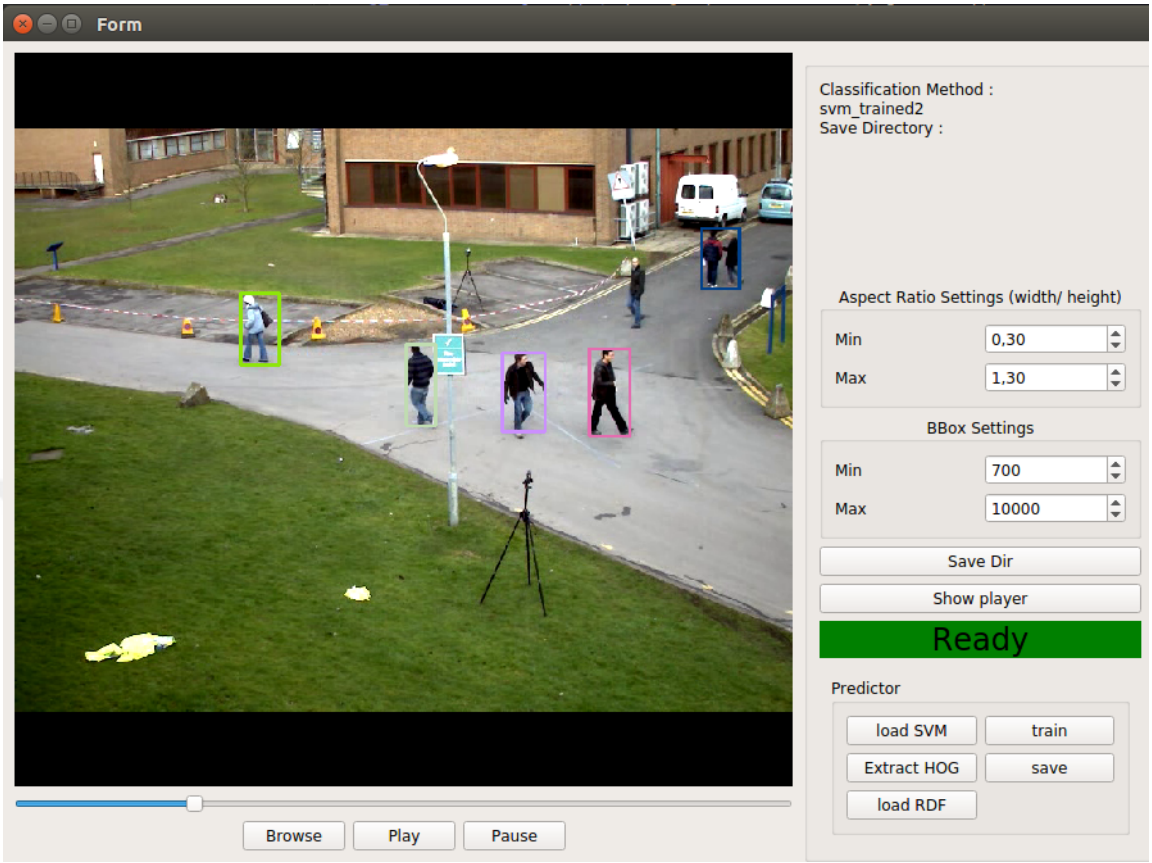


Figure 21: Data Extractor User Interface

- For some computers, RAM is limited and thus you are not able to use hole dataset for some cases. For these kind of situations, we added an option to load a fraction of the given data.

- To test implemented algorithms, we might be using the data from the different section of the same dataset. For these cases, we need to be able to skip some portion of data. By setting the number of images to skip, this module will skip from the start of the dataset while loading.

- Set color option allows you to set how images are loaded. It supports grayscale image loading and color image loading.

- Since we are using $L^*A^*B^*$ colorspace for our RDF training, we support $L^*A^*B^*$ color conversion.

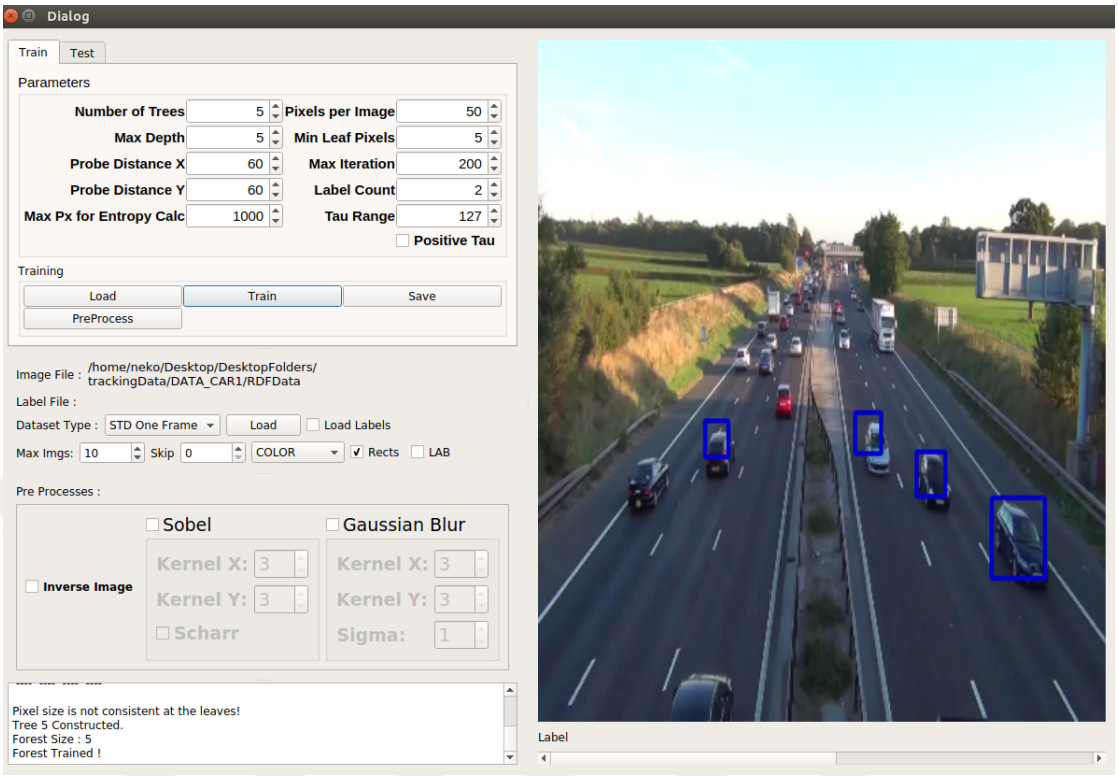


Figure 22: Random Decision Forest Training User Interface

- To see how the ground truth looks for debugging purposes we introduced an option to display ground truth on each frame.

2. Data Preprocessor. It is used to inverse image, do Sobel (Schar) [58] operators, and Gaussian Blur.

   - While loading MNIST dataset [59], we get images with white background and gray foreground colors. To make algorithm implementation easier we needed the colors be the other way around. Therefore, we introduced a processor that produces an inverse image.

   - To be able to do testing on edges in the images, we provided this Sobel processor that produces the image with all edges detected by Sobel (Schar) operator.

   - Gaussian Blur processor is provided to perform Gaussian blur on images with

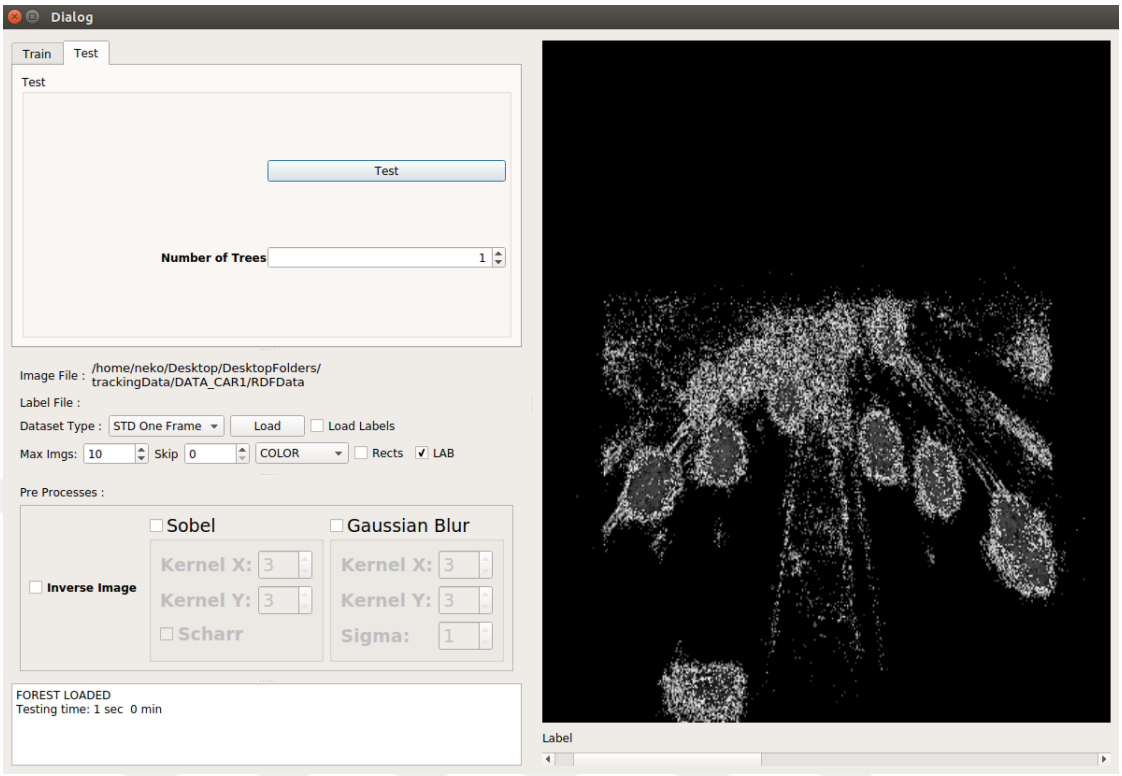


Figure 23: Random Decision Forest Testing User Interface

different kernel sizes.

3. Data Visualizer. It is composed of a label panel to display frames and a slider to go through each frame faster

4. Terminal. It is used to display training and test output for monitoring and debugging

3.4.4 Particle Filter Tracking Module

Particle Filter (PF) module is composed of three main parts that we use for tracking: Video Player, Predictor loader, and PF Settings

1. The video player is used for loading and playing videos. It contains VideoReader class which creates frame buffer from a video file, and a PF which is a video-processor. PF processes each frame and sends the result to the video player for

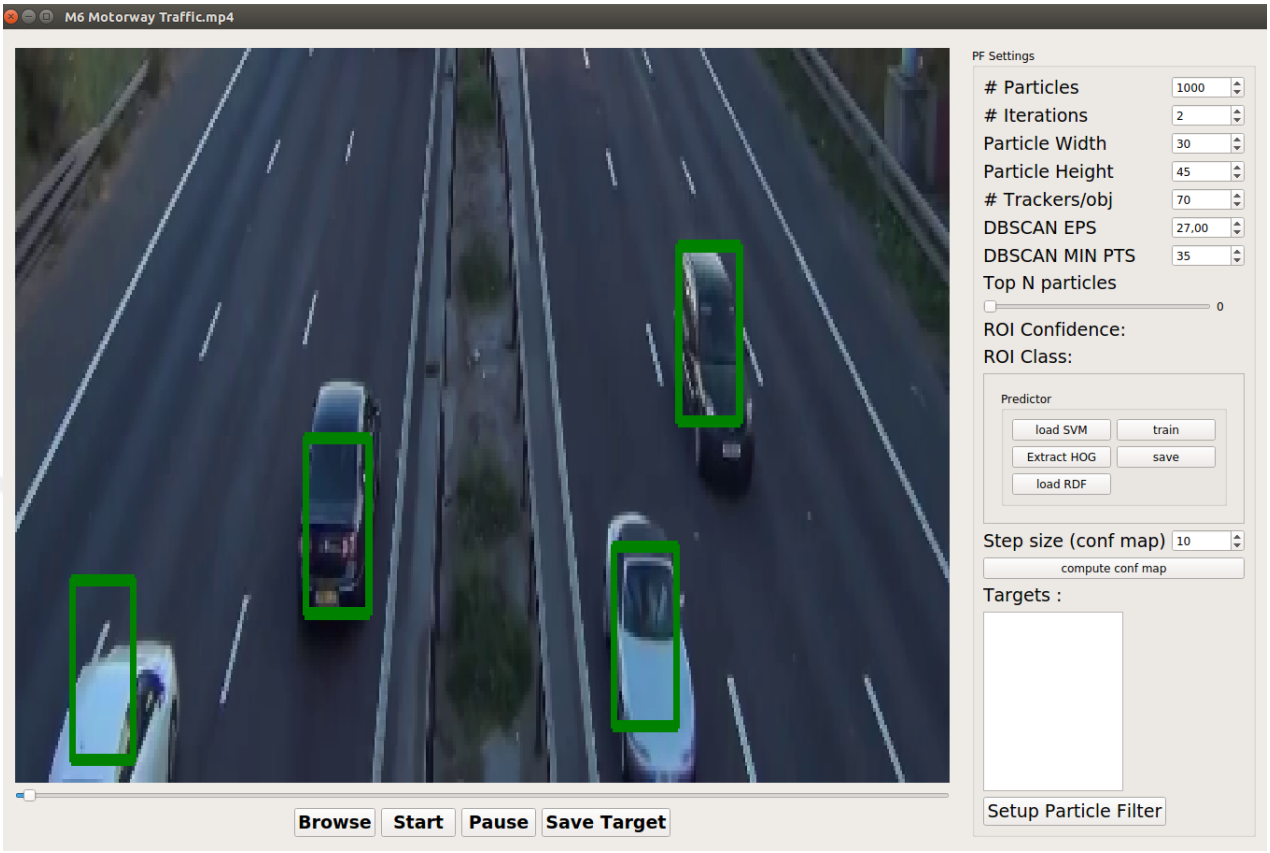


Figure 24: Particle Filter Tracking User Interface

displaying.

2. Predictor loader is used to load different predictors. We have:

   - SVM
   - HOG Extractor
   - RDF

   - SVM and HOG Extractor are used together. HOG Extractor is used to load training data and create HOG features. Pressing train, we train an SVM and save it. Later we can load trained SVM for prediction. However, these functions are not used for tracking purpose. It is used for data extraction, which is explained in the next section.

- RDF is our main concern in this section. It is used to load pre-trained RDF for PF.

3. PF Settings are used to set PF parameters which are shown in Table 1.

Table 1: Particle Filter Settings

Settings
of searchers
of search iterations
particle width
particle height
of trackers
DBSCAN epsilon
DBSCAN minimum points
"Top N particles" slider

- As explained in previous sections proposed PF is composed of fixed number of searcher particles and a varying number of tracking particles based on the number of vehicles being tracked. The number of searchers parameter sets fixed searcher particle count.

- For searching, randomly initialized particles are updated K time which is set by using the number of search iteration setter spin box.

- By setting particle width and particle height we are providing PF a prior knowledge about average with and height of particles (both searcher and tracker particles).

- Proposed method initializes M tracker particles per detected object and it can be set through setting the number of trackers parameter.

- DBSCAN [60] is a clustering algorithm that we use for clustering particles. To use this class epsilon value and minimum particle count should be set. To set these variables we have provided a specific parameter setter interface.

- For debugging purpose we provide a slider which sets a parameter N, which is used to display the top N searcher particles video stream.

CHAPTER IV

EXPERIMENTS

4.1 Setup

For the validation and parameter tuning of the random decision forest (RDF) we used RDF Trainer/Tester Module that we have implemented for this purpose. For particle filter (PF) validation we used PF Tracking Module. The details of their usage can be accessed in the Framework Section.

In order to validate the functionality and accuracy of the implemented Random Decision Forest (RDF), we tested the implementation on a toy problem. For this purpose, we decided to pick a well known MNIST [59] dataset of handwritten digits. It is composed of ten thousand test image samples and sixty thousand training image samples. For validation of our RDF implementation, we used pixel intensity difference of two randomly selected pixels as a feature for RDF decision node. Our input features (F_n) are θ_1^n, θ_2^n and image intensities. Thus our split function f_n became (Eq. 11). As shown in the results section (Figure 31), we were able to achieve image label accuracy of 98.24% for training data, and 99.41% for test data, by using (Figure 25) parameters for training. For this dataset pixel accuracy rate is relatively high, 86.80% for training data and 89.08% for test data. Since MNIST images are twenty eight by twenty eight we used thirty for both probe distances X and Y so that any given pixel will have a chance to be compared to all possible pixels in the image. We used tree depth of twenty four because of the RAM limitations of the laptop being used for testing. Since we used only foreground pixels for training, most of the pixels in any given image are background we computed that hundred pixels per image are enough sampling. After testing for various values of τ we found that hundred twenty seven

was a good choice, which makes sense given the fact that the difference between two pixels generally lays between ranges of plus and minus hundred twenty seven.

$$f_n(F_n) = I(x + \theta^n_{1x}, y + \theta^n_{1y}) - I(x + \theta^n_{2x}, y + \theta^n_{2y}) \tag{11}$$

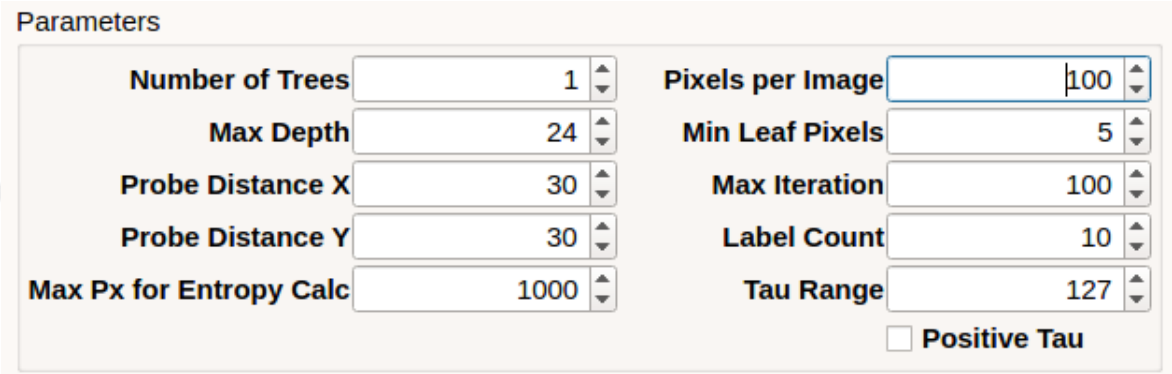


Figure 25: Random Decision Forest Training Params for MNIST

After we have validated that our RDF implementation is working perfectly, we tried to introduce new features. We decided to use Haar-like features for our split function. So, our new input features F_n became θ^n_1, H (H is the Haar-like feature) and image intensities and split function became (12).

$$f_n(F_n) = \sum_{i=0}^{h} \sum_{j=0}^{w} I(x + \theta^n_{1x} + i, y + \theta^n_{1y} + j) * H_n(i, j) \tag{12}$$

We used point, horizontal edges, vertical edges, horizontal lines, vertical lines and rectangle Haar-like features. The usage of these features increased detection accuracy. Same accuracy rates were achievable in a shorter tree depth, which can be seen from the results of the experiments in Table 2.

Next, as we validated our RDF implementation and saw an increase for MNIST dataset we decided to test the same algorithm and set-up for a different dataset. Thus, we used INRIA [61] person dataset. The test parameters are as shown in (Figure 26). As a result we obtained higher accuracy while using only Haar-like features with

respect to two pixel intensity difference and the mixture method. The test results are shown in (Table 3).

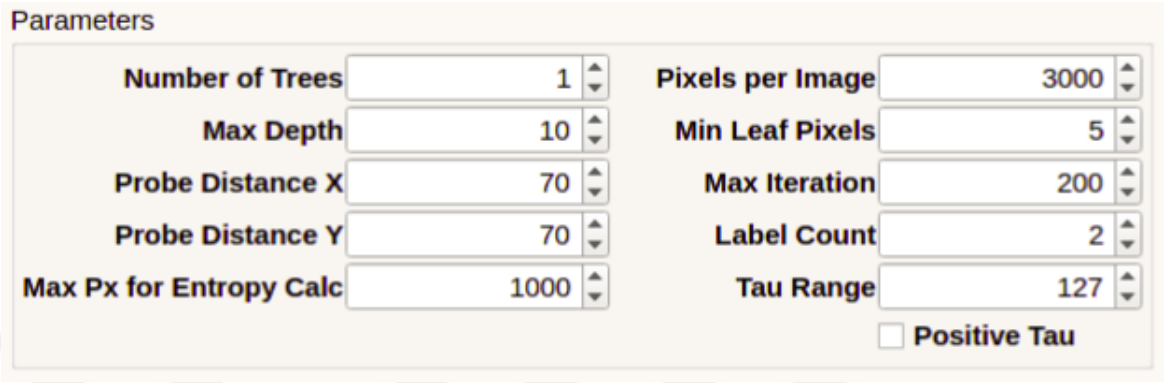


Figure 26: Random Decision Forest Training Params for INRIA

To test the implementation of particle filters we tested it using SVM trained on INRIA [61] person dataset. We used our HOG data extractor to extract HOG features from INRIA dataset. SVM was trained on extracted HOG features and used for weight assignment for particles. Later, using PF Tracking Module of our framework we tested the implemented PF algorithm. The verification of the algorithm was done based on how close single person on video frame was tracked visually.

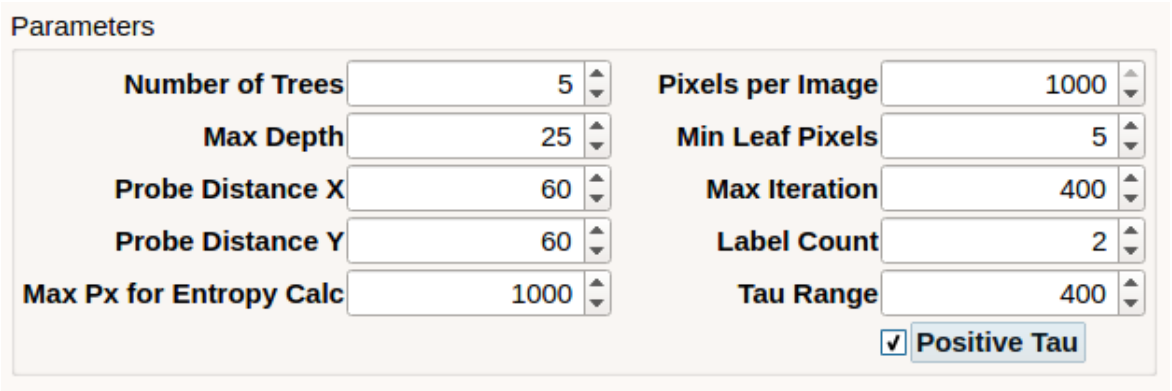


Figure 27: Random Decision Forest Training Params for Vehicle Detection

For the verification of the proposed system, we needed some car surveillance benchmark. However, since there is no annotated benchmark for multi-object vehicle detection on surveillance camera, we collected annotations using "YOLO darknet" [62] framework. The video stream that we have used is from "M6 motorway traffic" in the United Kingdom, which is thirty minutes in duration. The video stream is twenty-four frames per second. The data was sampled with a two frames per second sampling rate. Therefore, we were able to annotate five thousand frames. Half of the frames were used for RDF training. We would like to point that, the given data contains lots of noise and is not fully accurate since darknet framework does not give fully accurate annotations for the given video stream. We trained forest using parameters in (Figure 27). Since we are comparing our results with YOLO, all frames were resized to 416x416 which is the size YOLO uses for detection. Tests were conducted on $Grayscale$, RGB and $L^*A^*B^*$ color spaces. We trained RDF using five trees with a depth of twenty five and since we did not get any further improvements with our training and the test speed is being affected by the depth of each tree since we need more time to traverse each tree. After conducting numerous experiments, we found that probe distance sixty for both X and Y direction is a perfect. More than sixty was affecting training because the presence of neighboring vehicle was leading to confusion while learning. When less than sixty, randomly generated vectors are not long enough to use all vehicle pixels for training. Moreover, since trailer trucks and lorries are not detected properly, they impose big noise to the training of the forests.

Finally, as we completed validation of our Haar-RDF implementation on these toy problems, we decided to increase the number of Haar-like features and add some extra information to the leaf node. While tree structure is learned and pixels for each leaf node are calculated we used K-Means algorithm to cluster the displacement of each pixel from center of the object. By storing this information we convert our Haar-RDF to a Regression Forest. Therefore, modified Haar-RDF allows us to obtain both

detection and classification in one pass over the forest due to the new information stored in the leafs (posterior distribution and bounding box estimation).

The proposed method was tested on PETS09 pedestrian dataset. For the testing purpose we used five trees with depth of fifteen. Due to the average pedestrian dimensions we estimated probe distance X to be sixty and probe distance Y to be hundred. From each pedestrian we obtained six hundred random pixels for each tree. Due to the reasons explained in the previous experiments we set tau range to be between plus and minus hundred twenty seven. As for Haar-like features we used hundred fifteen features shown in Figure 14 and difference of two regions θ_1 and θ_2 away from pixel of interest are used. In order to have more precise description we used three different sizes of the given features: six by six, twelve by twelve, and twenty four by twenty four. This way we obtained three hundred forty eight different features.

For testing, we had ground truth only for PETS09 training dataset, which is composed of seven hundred ninety five frames. Therefore, we used six hundred of these images for training and hundred ninety five for testing. To test obtained results we used development-kit provided by multi object tracking challenge [63]. We were able to obtain **96.4%** precision rate on detection, which is comparably higher relative to the similar RDF algorithms. Sample regression output is shown in Figure 28. On the left image we displayed pixel voting results and on the right image we show computed bounding boxes after Mean-Shift is done on the voting results.

4.2 Result

In this section results for the experiments section are provided. Table 2 and Table 3 show the detection label accuracy for MNIST and INRIA datasets. Table 4 and Table 5 show the detection pixel accuracy for MNIST and INRIA datasets. These results are used for the illustration of added information for the detection by the Haar-like features. Average feature selection rates for MNIST and INRIA are displayed in

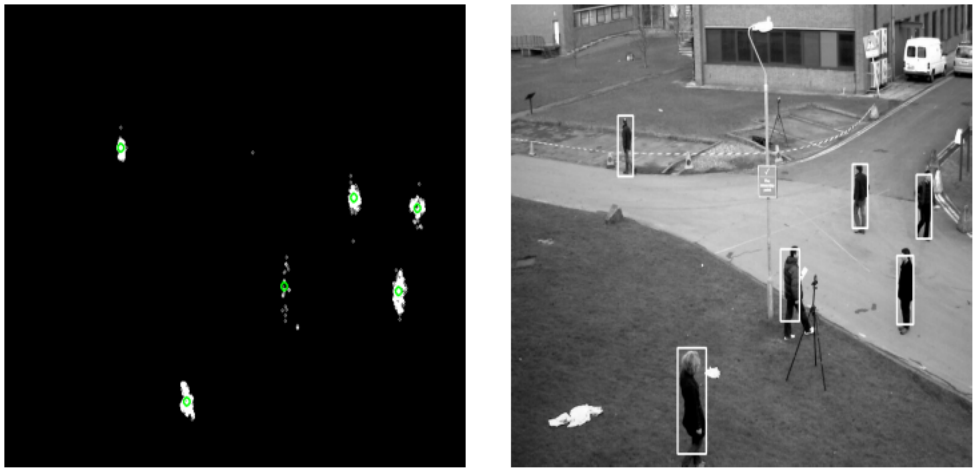


Figure 28: Regression Random Decision Forest voting result and computed bonding boxes after Mean-Shift clustering

Table 29 and Table 30 respectively. It can be observed that RDF mostly tends to pick single pixel difference feature more often.

Table 2: Detection accuracy for MNIST dataset obtained from a toy problem evaluating Haar-like feature contribution

	$single\_pixel$	$Haar-like$	$combined$
run_1	76.59 %	76.23 %	79.49 %
run_2	74.85 %	78.22 %	79.43 %
run_3	78.07 %	78.12 %	80.84 %
average	**76.50** %	**77.52** %	**79.92** %

The result for the complete label and pixel accuracy for the MNIST dataset is shown in Table 6. To display sample run output on the terminal while training, a screen shot is shown in Figure 32 and sample pixel voting is visualized in Figure 31. In the given image each color stands for a label. As we can see most of the pixels have voted for label being seven for this particular case.

Table 3: Detection accuracy for INRIA Person dataset obtained from a toy problem evaluating Haar-like feature contribution

	$single\_pixel$	$Haar-like$	$combined$
run_1	76.23 %	80.64 %	75.44 %
run_2	76.46 %	82.14 %	74.48 %
run_3	76.28 %	81.37 %	77.74 %
average	**76.32** %	**81.38** %	**75.89** %

Table 4: Detection pixel accuracy for MNIST dataset obtained from a toy problem evaluating Haar-like feature contribution

	$single\_pixel$	$Haar-like$	$combined$
run_1	34.62 %	24.14 %	29.36 %
run_2	31.16 %	27.22 %	34.43 %
run_3	32.86 %	26.43 %	29.86 %
average	**32.88** %	**28.93** %	**31.22** %

Table 5: Detection pixel accuracy for INRIA Person dataset obtained from a toy problem evaluating Haar-like feature contribution

	$single\_pixel$	$Haar-like$	$combined$
run_1	65.07 %	44.31 %	64.75 %
run_2	64.51 %	59.14 %	63.81 %
run_3	65.15 %	58.15 %	49.62 %
average	**64.91** %	**53.87** %	**59.39** %

Table 6: MNIST Accuracy

	$label\_accuracy$	$pixel\_accuracy$
$test\_data$	98.24%	86.80%
$training\_data$	99.41%	89.08%

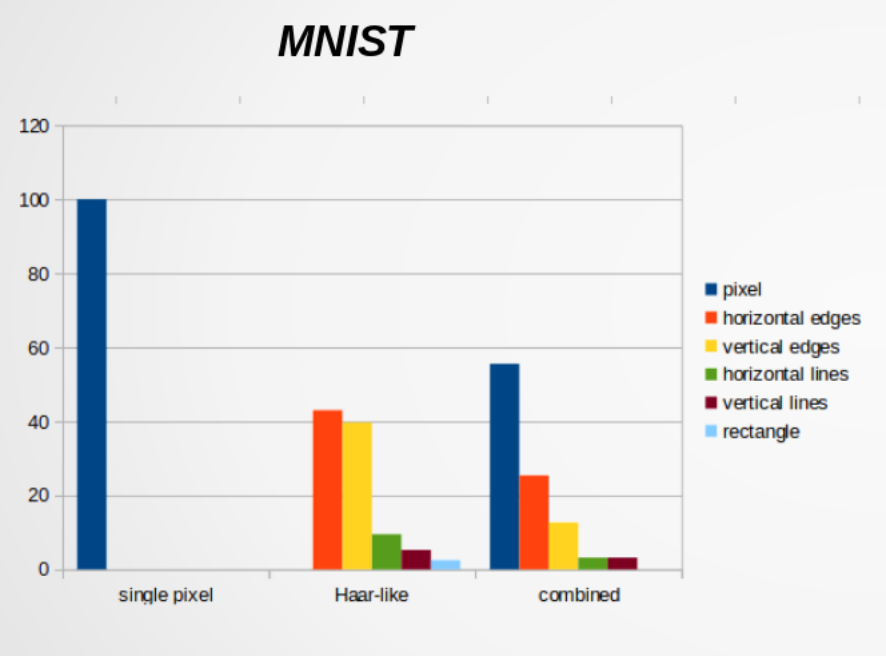


Figure 29: Average feature selection rate for MNIST

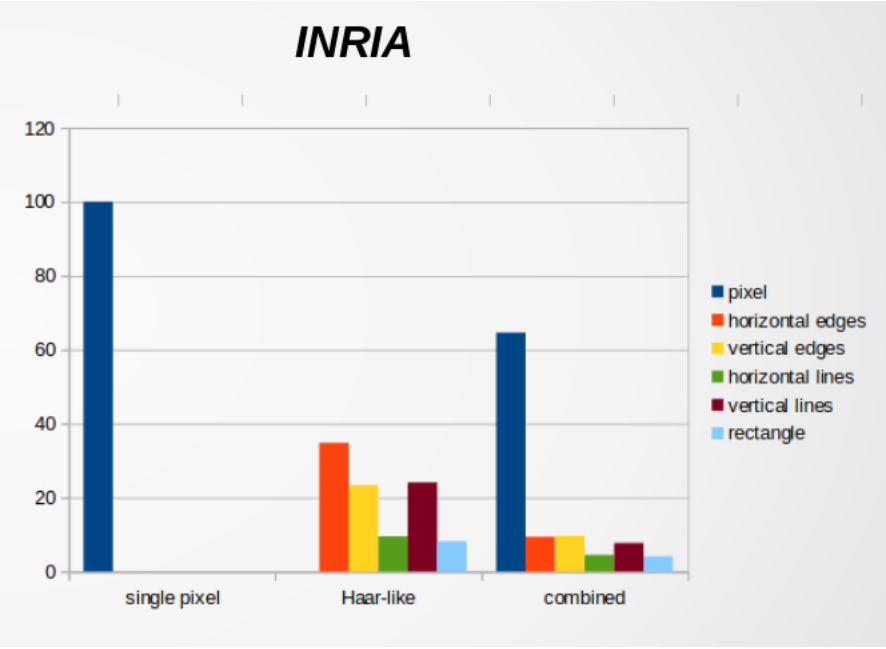


Figure 30: Average feature selection rate for INRIA

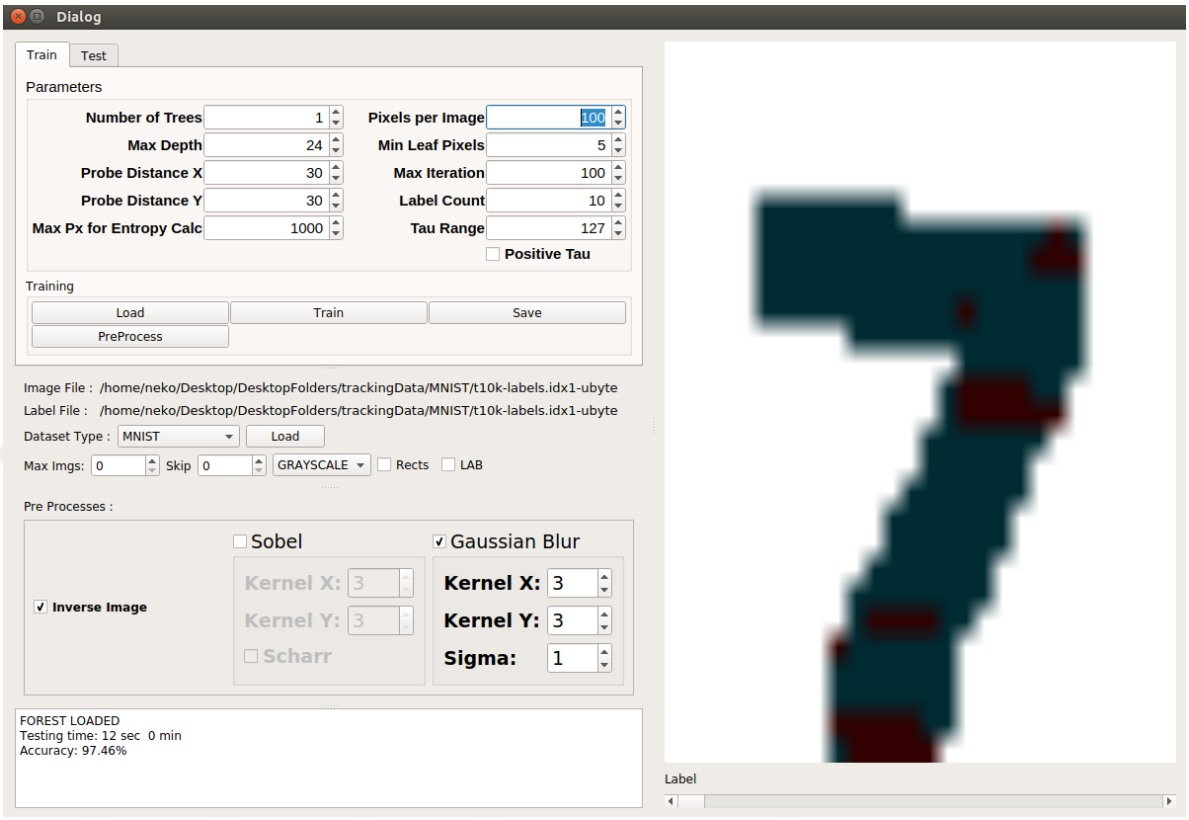


Figure 31: Random Decision Forest MNIST test result

```
<<<< DataSet >>>>
   images: 60000
   Images per Label:
      0 => 5923
      1 => 6742
      2 => 5958
      3 => 6131
      4 => 5842
      5 => 5421
      6 => 5918
      7 => 6265
      8 => 5851
      9 => 5949

Tree number 1 is being trained
Depth  Pixels  PxRatio  LCount  Impurity  ImpRat
0  3380547  100%  0  7783978  100%
1  3380547  100%  0  7783794  99.9976%
2  3380547  100%  0  7731513  99.326%
3  3380547  100%  0  7631278  98.0383%
4  3380547  100%  0  7514264  96.535%
5  3380547  100%  0  7383821            94.9 %
Training Time : 391 sec  /              6.52 min
```

Figure 32: Sample training output

Table 7: Average detection ratio in Vehicle Tracking

	Accuracy
YOLO	100%
Proposed_Method	92%

Table 8: Average number of frames processed per second for CPU and GPU in conventional computer

	FPS
GPU	30.3
CPU	3.8

CHAPTER V

CONCLUSION

There are lots of algorithms that have been developed for multi-object detection and tracking; particularly deep learning based algorithms that produce solid results, which, however, do not take into account the real-time processing time of the algorithms. As the number of cameras in the automated vehicle surveillance systems increases, the effective low computational cost algorithms become essential. For this reason, many researchers have developed alternative algorithms that are computationally cheaper. One of these techniques used in the literature is using Random Decision Forests (RDF), which is known for its balanced test-time performance both for speed and quality. Different variations include regression forests, Hough forests, cascaded random forest and etc.. Our work presented in this thesis, introduces a novel Haar-like Random Decision Forest. Haar-like features are introduced in the RDF split function which is a few orders of magnitude faster to evaluate. This new feature allows us to reach higher accuracy rates in shallower forest trees, which consequently make algorithms using RDF even faster. In order to test the usage of this new RDF, we implemented a vehicle tracking system that uses RDF and Particle Filters (PF). The nature of PF and RDF allows for fast computation. We use RDF to set particle probabilities for the future re-sampling stage of particles. Implementing them on GPU, facilitated us with a speed up of eight times faster than the implementation on CPU of a conventional computer. The proposed system is modular and any number of Haar-like features can be supported. It has been shown that as the diversity of Haar-like features increases, our system performs better for vehicle surveillance. With the proposed algorithm we were able to reach high precision rate ($96.4\%S$) for detection

on PETS9 pedestrian dataset. Moreover, we showed that compared to YOLO, one of the fastest state of the art deep-learning detection algorithms, in the conventional computer set up, our system generates comparable vehicle detection accuracy and real-time detection speed. The other contribution of this work is the introduction of a framework that facilitates algorithm training and testing. Additionally, it provides various modules for data extraction and data generation.

As future work, we are planning to extend our training for more Haar-like features with randomly generated dimensions and some other features. We are planning to implement RDF using different color spaces as well. Next step will be optimizing PF implementation and run particles on GPU and compare its performance with the state of the art tracking algorithms.

Bibliography

[1] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Discriminatively trained deformable part models, release 5." http://people.cs.uchicago.edu/ rbg/latent-release5/.

[2] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*, pp. 21–37, Springer, 2016.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

[6] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua, "Multiple object tracking using k-shortest paths optimization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 9, pp. 1806–1819, 2011.

[7] F. Fleuret, J. Berclaz, R. Lengagne, and P. Fua, "Multicamera people tracking with a probabilistic occupancy map," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 267–282, 2008.

[8] E. Yang, J. Gwak, and M. Jeon, "Conditional random field (crf)-boosting: Constructing a robust online hybrid boosting multiple object tracker facilitated by crf learning," *Sensors*, vol. 17, no. 3, p. 617, 2017.

[9] G. Pulford, "Taxonomy of multiple target tracking methods," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 152, no. 5, pp. 291–304, 2005.

[10] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik, "A real-time computer vision system for vehicle tracking and traffic surveillance," *Transportation Research Part C: Emerging Technologies*, vol. 6, no. 4, pp. 271–288, 1998.

[11] S. Tian, F. Yuan, and G.-S. Xia, "Multi-object tracking with inter-feedback between detection and tracking," *Neurocomputing*, vol. 171, pp. 768–780, 2016.

[12] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler, "Online multi-target tracking using recurrent neural networks," *arXiv preprint arXiv:1604.03635*, 2016.

[13] L. Wen, Z. Lei, S. Lyu, S. Z. Li, and M.-H. Yang, "Exploiting hierarchical dense structures on hypergraphs for multi-object tracking," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1983–1996, 2016.

[14] T. K. Ho, "Random decision forests," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1, pp. 278–282, IEEE, 1995.

[15] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–I, IEEE, 2001.

[16] S. Sivaraman and M. M. Trivedi, "A general active-learning framework for on-road vehicle recognition and tracking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 267–276, 2010.

[17] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *null*, p. 734, IEEE, 2003.

[18] P. Negri, X. Clady, S. M. Hanif, and L. Prevost, "A cascade of boosted generative and discriminative classifiers for vehicle detection," *EURASIP Journal on Advances in Signal Processing*, vol. 2008, p. 136, 2008.

[19] A. Barczak, F. Dadgostar, and C. Messom, "Real-time hand tracking based on non-invariant features," in *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE*, vol. 3, pp. 2192–2197, IEEE, 2005.

[20] J. Barreto, P. Menezes, and J. Dias, "Human-robot interaction based on haar-like features and eigenfaces," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 2, pp. 1888–1893, IEEE, 2004.

[21] K. Okuma, A. Taleghani, N. d. Freitas, J. J. Little, and D. G. Lowe, "A boosted particle filter: Multitarget detection and tracking," *Computer Vision-ECCV 2004*, pp. 28–39, 2004.

[22] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, "Hough forests for object detection, tracking, and action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 11, pp. 2188–2202, 2011.

[23] N. Gordon, B. Ristic, and S. Arulampalam, "Beyond the kalman filter: Particle filters for tracking applications," *Artech House, London*, vol. 830, 2004.

[24] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.

[25] C. Choi and H. I. Christensen, "Robust 3d visual tracking using particle filtering on the special euclidean group: A combined approach of keypoint and edge features," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 498–519, 2012.

[26] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[27] F. Baumann, A. Ehlers, K. Vogt, and B. Rosenhahn, "Cascaded random forest for fast object detection," in *Scandinavian Conference on Image Analysis*, pp. 131–142, Springer, 2013.

[28] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[29] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition.," in *Icml*, vol. 32, pp. 647–655, 2014.

[30] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2011.

[31] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[32] J. Yan, Z. Lei, L. Wen, and S. Z. Li, "The fastest deformable part model for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2497–2504, 2014.

[33] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Cascade object detection with deformable part models," in *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pp. 2241–2248, IEEE, 2010.

[34] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, ACM, 2014.

[35] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[37] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[38] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*, pp. 740–755, Springer, 2014.

[39] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[40] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[41] H. Jiang, S. Fels, and J. J. Little, "A linear programming approach for multiple object tracking," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8, IEEE, 2007.

[42] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE, 2008.

[43] A. A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu, "Multi-object tracking through simultaneous long occlusions and split-merge conditions," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 666–673, IEEE, 2006.

[44] A. Doucet, S. Godsill, and C. Andrieu, "On sequential monte carlo sampling methods for bayesian filtering," *Statistics and computing*, vol. 10, no. 3, pp. 197–208, 2000.

[45] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1515–1522, IEEE, 2009.

[46] C. Keskin, F. Kıraç, Y. E. Kara, and L. Akarun, "Real time hand pose estimation using depth sensors," in *Consumer Depth Cameras for Computer Vision*, pp. 119–137, Springer, 2013.

[47] J. Marin, D. Vázquez, A. M. López, J. Amores, and B. Leibe, "Random forests of local experts for pedestrian detection," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2592–2599, 2013.

[48] S. Schulter, C. Leistner, P. Wohlhart, P. M. Roth, and H. Bischof, "Accurate object detection with joint classification-regression random forests," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[49] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[50] H. Chouaib, O. R. Terrades, S. Tabbone, F. Cloppet, and N. Vincent, "Feature selection combining genetic algorithm and adaboost classifiers," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, IEEE, 2008.

[51] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics Springer, Berlin, 2001.

[52] M. Pollock, "Algorithms & computationally intensive inference reading group introduction to particle filtering discussion-notes," *Journal of the American Statistical Association*, 2010.

[53] S. Kim, N. Shephard, and S. Chib, "Stochastic volatility: likelihood inference and comparison with arch models," *The review of economic studies*, vol. 65, no. 3, pp. 361–393, 1998.

[54] M. Johannes and N. Polson, "Particle filtering," *Handbook of Financial Time Series*, pp. 1015–1029, 2009.

[55] Y. Salimpour and H. Soltanian-Zadeh, "Particle filtering of point processes observation with application on the modeling of visual cortex neural spiking activity," in *Neural Engineering, 2009. NER'09. 4th International IEEE/EMBS Conference on*, pp. 718–721, IEEE, 2009.

[56] P. M. Djuric and M. F. Bugallo, "Estimation of stochastic rate constants and tracking of species in biochemical networks with second-order reactions," in *Signal Processing Conference, 2009 17th European*, pp. 2308–2311, IEEE, 2009.

[57] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 2, pp. 28–31, IEEE, 2004.

[58] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[59] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[60] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[61] N. Dalal and B. Triggs, "Inria person dataset," *Online: http://pascal. inrialpes. fr/data/human*, 2005.

[62] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[63] L. Leal-Taixé, A. Milan, I. D. Reid, S. Roth, and K. Schindler, "Motchallenge 2015: Towards a benchmark for multi-target tracking," *CoRR*, vol. abs/1504.01942, 2015.

[64] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4705–4713, 2015.

[65] I. Szottka and M. Butenuth, "Advanced particle filtering for airborne vehicle tracking in urban areas," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 3, pp. 686–690, 2014.

[66] T. Gao, G. Li, S. Lian, and J. Zhang, "Tracking video objects with feature points based particle filtering," *Multimedia Tools and Applications*, vol. 58, no. 1, pp. 1–21, 2012.

[67] A. Criminisi, J. Shotton, E. Konukoglu, *et al.*, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends® in Computer Graphics and Vision*, vol. 7, no. 2–3, pp. 81–227, 2012.

VITA

Nekruzjon Maxudov completed his Bachelor of Science degree in Electrical-Electronics Engineering department at Ozyegin University, Istanbul, in 2014. His specialization was Multimedia and Signal Processing. He is pursuing his Master of Science in Computer Science department at Ozyegin University and his studies are concentrated on "Object Detection and Tracking". His masters thesis work was supported by the "TUBITAK 2215 - Graduate Scholarship Programme for International Students".