Hierarchical Bayesian Data Fusion Using Autoencoders

Yevgeniy Vladimirovich Reznichenko
Marquette University

HIERARCHICAL BAYESIAN DATA FUSION USING AUTOENCODERS

by

Yevgeniy V. Reznichenko, B.S.

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

August 2018

ABSTRACT
HIERARCHICAL BAYESIAN DATA FUSION
USING AUTOENCODERS

Yevgeniy V. Reznichenko, B.S.

Marquette University, 2018

In this thesis, a novel method for tracker fusion is proposed and evaluated for vision-based tracking. This work combines three distinct popular techniques into a recursive Bayesian estimation algorithm. First, semi supervised learning approaches are used to partition data and to train a deep neural network that is capable of capturing normal visual tracking operation and is able to detect anomalous data. We compare various methods by examining their respective receiver operating conditions (ROC) curves, which represent the trade off between specificity and sensitivity for various detection threshold levels. Next, we incorporate the trained neural networks into an existing data fusion algorithm to replace its observation weighing mechanism, which is based on the Mahalanobis distance. We evaluate different semi-supervised learning architectures to determine which is the best for our problem. We evaluated the proposed algorithm on the OTB-50 benchmark dataset and compared its performance to the performance of the constituent trackers as well as with previous fusion. Future work involving this proposed method is to be incorporated into an autonomous following unmanned aerial vehicle (UAV).

ACKNOWLEDGMENTS

Yevgeniy V. Reznichenko, B.S.

I would like to thank God and the saints, the Church, my family, friends, pastors, professors, colleagues and people around me without whom I would not have done this.

I would like to thank the Electrical and Computer Engineering department, the Graduate School and all of the Marquette University administration.

Finally, I would like to thank Dr. Medeiros for giving me the opportunity to do research together.

Table 1: Table of Notation

r, s	$\triangleq$	centroid of target.
t, u	$\triangleq$	height and width of target.
$x(t)$	$\triangleq$	True state. x_i and x_j represent the state vectors corresponding to two different trackers.
$y(t)$	$\triangleq$	Observed state.
A	$\triangleq$	State transition matrix.
C	$\triangleq$	State observation.
w	$\triangleq$	Process noise.
v	$\triangleq$	observation noise.
R_{ww}	$\triangleq$	Process noise covariance.
R_{vv}	$\triangleq$	Observation noise covariance.
Σ	$\triangleq$	Innovation covariance matrix with Σ_{ii} as the diagonal elements.
Ω	$\triangleq$	Mahalanobis distance.
ξ	$\triangleq$	Offset when penalization takes place for Mahalanobis weighting.
$d_{i,j}$	$\triangleq$	The Euclidean distance between x_i and x_j.
min_d	$\triangleq$	represents the smallest distance between tracker i and all of the other trackers.
w_d	$\triangleq$	Weight based on distance.
w_M	$\triangleq$	Weight based on Mahalanobis distance.
x_f	$\triangleq$	fused bounding box
$h(x)$	$\triangleq$	hidden representation function.
W	$\triangleq$	weight matrix.
ι	$\triangleq$	bias vector.
$\tilde{x}$	$\triangleq$	reconstruction from autoencoder.
N	$\triangleq$	number of total trackers.
τ	$\triangleq$	Threshold between outliers and inliers.
$b_m^{(n)}$	$\triangleq$	bounding box generated by the n-th tracker at frame f_m.
$\mathcal{F}$	$\triangleq$	represents the set of frames from a dataset. $\mathcal{F}^{(1)}$ is the training set. $\mathcal{F}^{(2)}$ is the testing set. $\mathcal{F}_{O^{(k)}}$ represents frame that just tracker just tracker k, is anomalous, all other trackers are with some range of the ground truth. $\mathcal{F}_S$ represents a set of frames where all trackers are within some range of the ground truth. $\mathcal{F}_{l^{(k)}}$ represents a set of frames where tracker n are within some range of the ground truth. $\mathcal{F}_{e^{(k)}}$ represents a set of frames where all trackers are within some range of the ground truth.
$R_{\sigma\sigma}$	$\triangleq$	Measurement noise covariance matrix of fusion KF.

Table 2: Table of Notation-Part 2

Φ	$\triangleq$	Total reconstruction error for input set of examples to network.
Ξ	$\triangleq$	input set of examples to network.
l	$\triangleq$	Layer of network.
$\tilde{f}_m$	$\triangleq$	vector corresponding to reconstruction of network.
f_m	$\triangleq$	feature vector corresponding to the concatenation of the outputs of all the Kalman filters.
L_i	$\triangleq$	the dimensionality of the l-th hidden layer.
M	$\triangleq$	Number of frames in sequence, m corresponds to a specific frame.
q	$\triangleq$	Encoded representation.
p	$\triangleq$	Decoded representation.
E	$\triangleq$	Kullback-Leibler divergence.
α	$\triangleq$	Activation function for a convolutional layer.
σ	$\triangleq$	Non-linear hyperbolic tangent activation function.
K	$\triangleq$	Number of k networks for each of the n trackers.
z	$\triangleq$	encoded representation.
ϵ	$\triangleq$	L2 regularization parameter.
ϱ	$\triangleq$	Reconstruction error.
P	$\triangleq$	Probability.
$\mathcal{P}$	$\triangleq$	Log likelihood score.
ρ	$\triangleq$	Parameters for autoencoder based score.
κ	$\triangleq$	Parameters for autoencoder based score that regulates speed of transition.
$\tilde{x}$	$\triangleq$	Corrupted input to network.
v	$\triangleq$	Number of neurons in layer l.
$k\ n_m^{(n)}$	$\triangleq$	State vector of tracker n for frame m.
$\bar{b}_m$	$\triangleq$	Ground truth.
θ	$\triangleq$	Weights for whole neural network.
ψ	$\triangleq$	Offset for maximum log likelihood.
ρ	$\triangleq$	Parameters for autoencoder-based weight.
Γ, Δ	$\triangleq$	The function parameters of the diagonal matrix for $R_{\sigma\sigma}$.
J	$\triangleq$	The intersection over union between a tracker and it's ground truth correct value.
ε_k	$\triangleq$	Stack to store reconstruction error for network k.
λ	$\triangleq$	Variable controlling how often the offset should update.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the current industrial, commercial and consumer market trends, it is evident that autonomous vehicles and semi-guided machines represent an active research area that is transitioning from theory to product. In this context, the majority of these systems is supported by vision-based tracking algorithms. Extensive and ambitious projects such as Amazon's drone delivery service [34], Uber [7], Tesla's self driving cars [19] and the first woman robot Sophia [22], exemplify just how rapidly robotics enter into our everyday lives. Unmanned Aerial Vehicles or UAVs are especially advantageous due to their relatively cheap cost and aerial nature. However, all of these products are not new, but rather they are the sum of hierarchical building blocks and incremental progress. Recent advancements in autonomous UAVs can be attributed to a few factors, including stronger and more compact computing, image processing and computer vision, machine learning and powerful sensors. Rather than being isolated developments, these advancements have all been intrinsically linked by growth in processing power for the onboard computer of these systems.

In this field, the problem of object following and its sub-problem of tracking are of great relevance. In fact, many state-of-the-art robotics applications require powerful and robust trackers [66, 28]. In tracking, there are many different approaches that have been proposed. Visual, global positioning systems (GPS) and infrared remain commonly used [5]. Visual tracking began first with template matching, a simple algorithm that has many limitations such as the simple model that does not update and the simple representation of the target. Afterwards, there was a transition to more sophisticated solutions such as Sift-Points [62]. Next, to address the issue with the object changing through time, early machine learning approaches based on algorithms such as support vector machines (SVM) [25], ensembling [58] and K-nearest neighbors (KNN) [35] were proposed. Finally, the past 12 years have seen the rise of deep neural network based approaches which have been revolutionary in the field of computer vision which includes the seminal Alexnet [43]. Previous tasks such as image classification have seen great strides, and helped introduce new tracking approaches. While state-of-art deep neural network (DNN) based trackers such as MDNet [61], SANet [20] and DCPF [31] effectively outclass most of the previous approaches, unfortunately, they remain too slow for real-time application. With new research in skip connections [26], attention models [81], dilated/atrous convolutions [80], reinforcement learning [49] and capsule networks [72], it remains probable that improvements are on the horizon. While all of the approaches provide significant improvements, we address the fundamental issue of robustness by proposing an algorithm that can fuse information from various single object vision trackers to provide robustness at the cost of computational complexity. Our definition of robustness includes both generality, such that the network can operate well in many different scenarios and a tolerance to faults. One of the fundamental advantages of the deep learning revolution has been to introduce deep networks, that with sufficient data, can construct regression models that model complex unknown mathematical behaviors. Our aim is to leverage this tool to learn the operation of any type of tracker to model normal operation. Once our network can model normal operation, we use this information to act as a weight in a fusion step.

Because robustness is so important in critical systems, it has received a lot of attention. In autonomous vehicles, any sort of error can be extremely costly, and therefore many systems include redundancies in order to improve performance. This idea is the primary motivation behind the sub-field of sensor fusion. Sensor fusion rose to prominence with the invention of the linear Kalman filter (KF) in the 1960s [10]. However, the linear KF has limitations including the Markov assumption [10] and it assumes a linear motion. Subsequent innovations such as the Extended and Unscented Kalman filter [10] have tried to address this issue with varying levels of success. Further improvements that sought to incorporate prior information introduced other Bayesian methods such as Particle filters [17]. As in tracking, machine learning also became prominent in sensor fusion and in combining classifiers. SVMs [48], Naive Bayes [45], Majority voting [45] and Adaboost [1] have been powerful techniques that are well known and also well understood. With the deep learning revolution, deep neural networks have been used to fuse results from different sensors [13], learn values for a deep Kalman filter [40] and fuse different types of information in a siamese network [4]. We build upon the work of Echeverri et al. [18] which already uses Kalman filters and simpler machine learning techniques by applying deep networks to learn powerful representations of the intrinsic characteristics of properly functioning trackers. We integrate deep Bayesian autoencoders into this framework to improve performance.

To train our networks, we first collect data on the OTB-100 dataset [84]. This is a dataset that contains 100 distinct images sequences with a moving target. For each of these videos, algorithms are given the initial location of the object. After the trackers run through these video sequences, a MATLAB script evaluates all trackers' performance on these videos. By running our tracker ensemble on this dataset we can collect the results of various trackers. Because this dataset contains the ground truth, the results can be split into various partitions. We look at different subsets to find representative data that we use to train networks that recognize distinct anomalies. We formulate the score as a maximum likelihood probability [45] similar to a naive Bayes approach to penalize anomalies. Later, we add this feature to the ensemble and run the sensor fusion algorithm on the benchmark. After evaluating this new algorithm on the benchmark, we look at its performance on a test and training set of videos.

Our final goal is to deploy the algorithm on a UAV drone and use the Matrice 100, DJI SDK [71] and add PID controls based in part on the work in [66] to have the drone follow a target. We would qualitatively compare performance to available commercial applications; namely the DJI Mavic. To compare, we would first look at the previous AR-parrot version described in [18] to determine whether our new solution provided any significant improvements. Our future tests would be to perform a quantitative analysis by examining the number of frames the drone can successfully follow a target for 3 different scenarios/environments. This includes outdoors, indoors and stationary. Here we could measure success by how many consequent frames the drone can follow a target and compare that to its competitor.

1.1 Contributions

The contribution of this thesis is the creation of a Bayesian Autoencoder Maximum A Posteriori Data Fusion framework (BAMAPDF). The purpose of this research is to improve upon the Hierarchical Bayesian Data Fusion (HABDF) algorithm developed in [18]. More specifically, the contributions are as follows:

1. Acquire training data from different trackers, partitioning and properly scaling data.
2. Explore different machine learning approaches to detect anomalies.
3. Integrate the trained models into the tracker ensemble to improve robustness.

The thesis is organized as follows. Chapter 2 introduces ideas that influenced our work as literature review. In Chapter 3, we look at previous work that this thesis builds upon. In Chapter 4, we evaluate the HABDF framework,present our data partitioning method and evaluate various machine learning based anomaly detection methods. In Chapter 5, we explain how we used the outlier detection results to modify the initial sensor fusion approach. In our conclusion, we summarize our work, and discuss future research on how to integrate the proposed method on a following UAV.

CHAPTER 2

LITERATURE REVIEW

This chapter is structured into four sections. The first section discusses related work on vision-based target tracking. Secondly, we introduce the proposed data fusion approach as the main contribution. Next, we look at related work on anomaly detection and introduce our choice of autoencoders. Lastly, we look at work related to UAVs and object following as a real-life application to our modified algorithm.

2.1 Tracking

Given an initial video frame and a bounding box that delimits an object of interest in that frame, the purpose of a single-target tracking algorithm is to follow this object through subsequent frames without being told the new bounding box that encompasses the object. This is relevant in many applications such as surveillance and autonomous driving where one needs to keep track of a single object over consecutive frames. While the problem sounds simple to a human, it is difficult for a machine due to the uncertain nature of this problem. Changes in lighting are trivial to us, but to a machine this change in the mathematical representation of the whole image is difficult to handle without a mathematical representation of this change and the conditions of this change. Due to the very open-ended and multifaceted nature of this problem, a myriad of different trackers has been developed.

Trackers such as TLD [35], for example, use a nearest neighbor based approach to address the issue of tracking an object between frames. State-of-the-art trackers such as GOTURN [27], use deep convolutional neural networks to track objects in a search space. Due to their inherent design characteristics, the performance of these trackers differ with respect to issues such as occlusion, illumination, motion, deformation, blur and rotation. As such, different trackers have been found to perform better depending on the scenario. With this in mind, while some trackers show overall better performance in standardized datasets, there are instances in which these trackers are outperformed by less sophisticated methods due to the fact that they do not handle a particular situation well. As it stands now, the tracking problem is still a largely open research topic.

2.1.1 Sensor Fusion

To generate more robust predictions, typically a common approach is to combine results from multiple sources. In fact, certain trackers are just combinations of a large amount of weaker trackers that work together [88]. Sensor fusion has been an intense area of research in controls and electrical engineering. The idea of adaptive fusion began in the 1960s with the introduction of the Kalman filter (KF). In the 1990s, the approach became more widespread with the development of variations of the initial algorithms based, for example, on the extended KF or the unscented KF [76]. Particle filters (PF) [23] and fuzzy logic [8] have also recently gained popularity. Some preliminary work has even been done on combining deep neural networks and Kalman Filters [40]. Each of these methods assumes some sort of prior knowledge about the trajectory of what is being tracked. For single-object-tracking, this is generally a valid assumption given the relatively locally linear nature of the motion of most objects when observed at reasonable frame rates. Our work is most similar to the work done by Bailer [3] and Biresaw [6], which use a hierarchical state fusion interpretation. However, we use a different data fusion approach than Biresaw, and our calculations are done with a Bayesian framework in contrast to the work done by

Bailer. Additionally, similar Bayesian frameworks were proposed by Yang [86] to perform multimodal tracking for healthcare applications with the use of different weighting schemes. One of the most common modern approaches to data fusion is the use of machine learning. These methods are powerful but are challenging to implement practically due to their reliance on large amounts of training data. The approach presented in this paper mitigates these issues by using an adaptive Bayesian model that adapts its behavior based on the performance of the trackers and by using a semi supervised approach that reduces the amount of training data needed. A hierarchical Bayesian data fusion approach requires only that the user provides weights to the trackers as a tuning parameter and a motion model which can be assumed to be linear.

2.2 Anomaly Detection

On the topic of anomaly detection, a myriad of methods have been proposed. Recently, the development of machine learning has allowed for complex rules to be derived to quantify errors. Common approaches include SVMs [64], Decision Trees [78], Naive Bayes [2] and Adaboost [29]. However, all of these methods are supervised and require labeled positive and negative data. While attempts have been made for unsupervised approaches such as the One-class SVM [79], results have been mixed. Other unsupervised approaches such as K-means have also been proposed, but unfortunately, this algorithm generates many classes and suffers in performance with large-dimensional data. Recently, deep learning has made tremendous strides in dealing with the issue of high dimensional data. In particular, autoencoders have been used as a way to reduce the dimensionality of data in a manner similar to PCA [69] by learning non-linear transformations.

Data fusion for object tracking has been explored in great detail in works such as [6, 11, 3]. The method of using confidence scores with a majority vote for data fusion was first proposed in Echeverri et al. [18] and later evaluated in this thesis. That method, referred to as Hierarchical Adaptive Bayesian Data Fusion (HABDF), has the advantage of being computationally inexpensive. However, one weakness of that approach is its susceptibility to anomalous tracker outputs. In particular, this is because the confidence score uses a Mahalanobis score to determine whether the tracker is an outlier based on [67]. This is problematic because the underlying assumption is that object motion is linear and noise is Gaussian. For tracking, this assumption does not always hold due to the non-linear nature of object motion in the wild. Therefore, the ability to handle situations where this assumption fails is important to improving performance.

Detecting outliers in a tracker is difficult because if a tracker knew when it was wrong, it would be able to self-correct preemptively and not make the mistake in the first place. A robust outlier detection mechanism is of particular interest for vision-based trackers. Penalizing anomalies is commonly used in tracker ensembles and sensor fusion. Normally, when the algorithm decides that a tracker is lost, it might discard the results or apply a smaller amount to that trackers result so its effect will be trivial or ignored. Better anomaly detection in each of these approaches would naturally lead to better results. One successful approach to determine that a tracker is lost was proposed in [83]. Unfortunately, that method is only applicable to trackers that employ correlation filters because it is dependent on the distribution of the correlation map generated by the network.

Autoencoders have shown great potential as a tool for anomaly detection [90, 50, 79]. However, to our knowledge no work has explored their use as a weighing mechanism for tracker ensembles. The methods proposed in [33, 38] are perhaps the closest to our work, albeit they were applied to the different problems of monitoring wind turbines and electrocardiograms. Our method builds on the works proposed in [33, 38] by building feature vectors consisting of several estimates of the positions and velocities of the target, which are generated by Kalman filters that use the ouputs of the individual trackers as observations. In addition, these feature vectors are constructed using consecutive image frames, thereby further incorporating the temporal relationships between the outputs generated by trackers.

2.3 Unmanned Aerial Vehicles

Unmanned UAVs or drones have recently captured the public's attention. Follow-me UAVs are an especially interesting area of research due to their application in the film industry. These are UAVs that receive a specified target and then attempt to follow the target with the camera focused on that target and controlled so that the target is in the center of the image. Rather than having someone control a camera to follow a scene an automated drone would decrease costs and lead to more freedom for artistic expression. Most higher end UAV platforms have some sort of onboard program to accomplish this task. In general, this is usually accomplished in one of two ways; using GPS [16]/Ground Station Control [65] or using recognition/tracking [66]. Using a ground station requires that the object of interest has some of device that allows the UAV to triangulate to the objects location. However, this incurs the issue that the device is intrusive. For very close following, the GPS is unreliable unless a very expensive option is chosen [54]. Using a ground station that relies on other signal types such as WiFi is possible but leads to issues associated with latency. An image based following approach has been accepted as the preferred method when dealing with smaller distances [54]. In particular, the work by Pestana [66] was instrumental in developing the original autonomous following UAVs in [18]. Furthermore, in addition to attempting to keep the target at the center of the image using its centroid position (*r*,*s*), the UAV also used the target's relative scale variations, based on *t* and *u*, to keep a constant distance from the target. Essentially, the drone would follow a target within a fixed distance, attempting to maneuver so that the target remains in the center of the video frame.

Our goal in this thesis is to improving our tracking performance by adding robustness via a more powerful fusion approach. To do this, we frame our fusion trust mechanism as an anomaly detection/anomaly score problem.

CHAPTER 3

BACKGROUND

Here, we introduce Hierarchical Bayesian Data Fusion (HABDF), the algorithm this thesis seeks to improve. We also evaluate this method and set it as reference to our approach. Next, we explain the theoretical foundations of autoencoders, which act as our main tool in improving HABDF.

3.1 Hierarchical Bayesian Data Fusion for Target Tracking

HABDF is a variation of the mixture of experts framework [87]. The main difference is that the gate is substituted with a Bayesian approach. Each separate tracker s_j acts as an "expert" asynchronously when it is run through a Kalman filter. The motion and observation models are given by

$$x(t) = Ax(t-1) + w(t) \tag{3.1}$$

$$y(t) = Cx(t) + v(t), \tag{3.2}$$

where x is the state vector and y is the observation vector. Eq. 3.1 represents the system dynamics with A representing the transition matrix, B being the control matrix and w modeling process noise. In Eq. 3.2, C is the observation matrix, and v is the measurement noise. Both of the noises are assumed to be white and Gaussian with variances R_{ww} and R_{vv}. HABDF uses two sources of information to penalize detectors and to vote on a global output. The first mechanism through which the framework assigns weights to each of the detectors is based on the Mahalanobis distances (MD) [53] of the observations, where μ is equal to prediction, and Σ is the covariance.

$$\Omega(y) = \sqrt{(y-\mu)^T \Sigma^{-1} (y-\mu)}, \tag{3.3}$$

As shown by Pinho [67] the MD can be approximated by

$$\Omega(y) = \sum_{i=1}^{N} \left(\frac{(y_i - \mu_i)^2}{\Sigma_{ii}} \right), \tag{3.4}$$

where y_i and μ_i are the elements of y and μ and Σ_{ii} are the diagonal elements of the innovation covariance matrix Σ.

Rather than using the MD values directly as weights in our framework, in order to soften transitions as the performance of the individual trackers fluctuates, a sigmoid function is employed

$$w_M = \frac{1}{1 + e^{(-\Omega(y) + \xi)}}, \tag{3.5}$$

where ξ is a value chosen based on the χ^2 number of degrees of freedom of the system and the desired confidence level. This step takes advantage of the Bayesian framework but rather than using those statistics to correct the tracker as done in [51, 46], here they are applied as weights in a voting scheme. This generates a score that penalizes trackers for being far away to the other nearest tracker.

The other mechanism involved in the assignment of weights to the outputs of the individual trackers is the majority voting scheme based on the pairwise Euclidean distances between the various trackers. Let x_i and x_j represent the state vectors corresponding to two different trackers. Let the Euclidean distance between x_i and x_j be

$$d_{i,j} = ||x_i - x_j||. \tag{3.6}$$

Then, min_d represents the smallest distance between tracker i and all of the other trackers in the framework.

$$min_d = \min_{\substack{j=1,2,\ldots,n \\ j \neq i}} (d_{i,j}). \tag{3.7}$$

Again a softening mechanism is applied to avoid abrupt changes in the tracker operation. In this case, the method chosen was a hyperbolic tangent function

$$w_d = \omega_0 + \omega(1 + \tanh(min_d - \lambda)), \tag{3.8}$$

where ω_0 is the minimum value and λ represents the minimum required for the penalization to take place.

The filtered outputs of all the trackers, bounding boxes x_j, are provided as inputs to another KF. This acts as the fusion center. The fusion center adapts itself to changes in the performance of individual trackers after each new measurement is collected by updating its measurement noise covariance according to

$$R_{\sigma\sigma}(w_d, w_M) = \Gamma w_d + \Delta w_M, \tag{3.9}$$

where $\Gamma = diag(\gamma_1, \gamma_2, \cdots, \gamma_n)$, $\Delta = diag(\delta_1, \delta_2, \cdots, \delta_n)$, and $diag(.)$ represents a diagonal matrix whose elements are the function parameters. γ_i and δ_i are set to 1 if there is no a priori knowledge of the system, but they can be adjusted individually if there is prior information about expected tracker performance. That is, the majority voting weight w_d and the MD weight w_M are used by the global tracker to update $R_{\sigma\sigma}$, which is then used in the global correction stage of the Kalman filter to generate the fused bounding box x_f. Eq. 3.9 allows the Kalman filter to trust less in measurements that have lower weights. The Kalman filter for the fusion center is essentially identical to those applied to the individual trackers (Eqs. 3.1, 3.2) but the observation matrix C reflects the fact that the observations are given by the outputs of the n trackers. Algorithm 1 summarizes the HABDF algorithm.

Algorithm 1 HABDF

```
Input: Set of $n$ trackers $s_j \in \mathbb{S}$, initial bounding box $x_0$, set $V$ of images
Output: Bounding box $s_f$ representing the fused output
 1: Initialize all trackers $s_j$ with $x_0$.
 2: Initialize Kalman filter for each algorithm implementation $s_j$
 3: Initialize Kalman filter for fused data model
 4: while $V$ has new images do
 5:     Load new image
 6:     for Each tracker $s_j \in S$ do
 7:         Generate bounding box $x_j$ for each tracker $s_j$
 8:         Apply Kalman filter (Eq. 3.1,3.2) to $x_j$
 9:         Compute Mahalanobis Distance weight $w_M$
10:     end for
11:     Wait for all trackers $s_j$
12:     Apply majority voting to find $w_d$
13:     Calculate $R_{\sigma\sigma}$ according to Eq. (3.9)
14:     Apply Kalman filter (Eq. 3.1,3.2) using $R_{\sigma\sigma}$ as the observation covariance to generate
            the global estimate $x_f$
15: end while
```

3.1.1 Additions to HABDF

In this thesis, we added the tracker GOTURN [27] to the ensemble. This tracker was considered a state-of-the-art real-time neural network based tracker at the time we started. To work as a tracker on a video benchmark, modifications had to be made. Because the original algorithm used asynchronous calculations for maximum speed, so that calculation speed depended only on the fastest tracker, the algorithm could skip frames. To rectify this, the algorithm was modified to be synchronized to the slowest tracker. Although this approach penalized the algorithm's speed, it could still operate in real-time. Additionally, accuracy and robustness was improved. By incorporating additional locks, concurrency issues in the threading were addressed.

3.2 Autoencoder

A important current research topic is the problem of outlier detection. Outlier detection, fault detection, anomaly detection are used interchangeably and refer to the concept of detecting when operation stops being “normal”. In power transmission this would be when current through a line drastically increases, commonly known as a three phase fault. When this is the case, we can see that this is happening with sensors and normally a relay is triggered to stop the current from flowing and damaging the transmission line. Other faults can occur in different types of systems from different application domains including raw vibration signals [89], turbines [33], altitude estimation [24] and big data [52]. This concept has many applications in fields of research including: tracking, big data, object detection and machine learning. Additionally, it is also an important standalone topic in areas such as aviation [32]. In object tracking, methods that rely on Bayesian estimation are not robust to anomalous data, especially since these methods use prior data to make an estimate. If a previous estimate is poor it can throw off a tracker, which will remain lost and hurt the final estimate. To improve outlier detection, several methods have been proposed [57]. One of the most promising solutions seems to be the machine learning approach due to the unique power of machine learning to learn non-linear abstractions [77]. One route of machine learning that seems to be promising is the use of autoencoders. Autoencoders are neural networks that model the target function input = output, and by doing so they learn a model for the latent space of expected data [42].

For any input example vector or matrix $x \in \mathbb{R}^n$, we can generate a hidden representation $h(x) \in \mathbb{R}^m$ by using a non-linear activation function applied to every component. We chose to use the hyperbolic tangent because it provided the best results experimentally, but many other options are available. With W as the weight matrix, and ι representing the bias vector we can formulate the encoder:

$$h(x) = f(W_1 x + \iota_1), \tag{3.10}$$

where the activation function $f(z)$ is equal to the hyperbolic tangent,

$$f(z) = tanh(z), \tag{3.11}$$

The decoder of the autoencoder then maps the hidden representation to the reconstruction $\tilde{x} \in \mathbb{R}^n$:

$$\tilde{x} = f(W_2 x + \iota_2), \tag{3.12}$$

Normally, multiple layers are stacked in the encoder and decoder in a descending manner to force the network to learn a latent representation of our data. Training the autoencoder is then done to find the parameters that minimize the mean squared error. With an input set of examples Ξ which can be defined as :

$$\Phi(\theta) = \sum_{x \in \Xi} \|x - \tilde{x}\|^2 . \tag{3.13}$$

Gaussian noise is usually added during training to the input to improve performance the networks performance by minimizing overfitting. Noise is added by corrupting the input x into corrupted input $\acute{x}|x \sim \mathbb{N}(x, \sigma^2 I)$.

This added benefit assists the network in learning a better latent representation[56]. Stochastic gradient descent is generally used due to find weights that minimize the mean squared error through backpropogation, but there are other options available such as Adam or RMSprop. In our work, we generally used RMSprop [70].

By taking advantage of the inherent ability of these neural networks to learn what governs "normal operation", it is then possible to extrapolate what is "unexpected" data. There are many methods that take advantage of this model and most of them use either the dimensionality reduction or the reconstruction error to detect outliers [50]. We use the reconstruction error as a threshold which can defined be defined as

$$\varrho = \|x - \tilde{x}\|^2 \tag{3.14}$$

3.2.1 Variational Autoencoder

To better model the probability model associated with many different types of datasets, further advances in machine learning have led to the creation of variational autoencoders [37]. The variational autoencoder uses hidden layers to learn latent representations in a manner similar to regular autoencoders. However, the difference is that the bottleneck layer encodes to a Gaussian probability density,

$$q_\theta(z|x) \sim \mathbb{N}(\eta_z, \zeta_z), \tag{3.15}$$

Where η_z and ζ_z are the mean and standard deviation of the distribution, respectively. Our encoded value is now z. To decode z the decoder outputs the parameter associated with each probability distribution of the data,

$$p_\phi(x|z) \sim \mathbb{N}(\eta_z, \zeta_z). \tag{3.16}$$

We can measure the loss function as a function of the information lost associated with the decoder as the sum of the reconstruction error of the representation or the mean squared error 3.13 and the Kullback-Leibler divergence between the encoder described in equation. 3.15 and a Gaussian distribution (normally mean zero and variance one).

$$\Phi_i = -E_{z \sim q_\phi(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||\mathbb{N}(0,1)), \tag{3.17}$$

Where the Kullback-Leibler divergence is defined as

$$E\left[\log \frac{q_\theta(z|x) \sim \mathbb{N}(\eta_z, \zeta_z)}{p_\phi(x|z)}\right]. \tag{3.18}$$

The Kullback-Leibler divergence attempts to force the latent distribution by penalizing the bottleneck distribution to fit the normal distribution. $-E_{z\sim q_{\phi}(z|x_i)}[\log p_{\phi}(x_i|z)]$ generally is approximated as the mean-squared error.

This is particularly relevant to our application because we predict that the variational autoencoder has the smallest possible reconstruction error for the good data, and that mapping occurs in such a manner that prevents overfitting by encouraging dispersion of the latent representation.

3.2.2 Convolutional Autoencoders

To assist our network in learning the temporal dependencies we also explored using 1D convolution layers in a manner similar to Wavenet [80]. This changes our neurons to become the convolution of the input and weight matrix from the previous layer. As expressed in [38], we define our network activations α for layer l for all v neurons as

$$\alpha(f^l) = \sum_{i=l}^{N_{l-1}} \sigma(W_{iv}^{l-1} * f_i^{l-1}) + \iota_v^l, \tag{3.19}$$

where "$*$" is the 1-dimensional convolution operator and $\sigma(\cdot)$ is the non-linear hyperbolic tangent activation function. An important consequence is that because of the convolution operator, our hidden layers are 2D. The last layer of a convolutional autoencoders is usually a flatten layer that brings the shape back to 1D.

In a similar manner, a 2D autoencoder can also be used to convolve 2D representations of data by modifying 3.19 to use higher dimensional tensors. Most image based applications use convolutional neural networks to achieve state-of-the-art results [15].

3.2.3 Apples and Oranges Example

To better illustrate how autoencoders can be used to detect anomalies we present the example of apples and oranges based in part on the work in Imagenet [44]. Imagenet is a dataset of over a million photos of different classes. One of the most well known neural networks used to classify different images and accomplish this task is VGG [73]. We base our encoder and decoder architecture heavily on the first 4 layers of VGG with one low-dimensional layer in the middle. As seen in the work by Dias [15], the network is capable of capturing representations of a complex class into a latent space. By taking advantage of Imagenet, we first acquired 100 images of the apple class and 100 images of the orange class. Next, to give our network more training examples we used the method detailed in [44] to generate more data because Imagenet does have many photos only a small minority are apples. Afterwards, we built a neural network based heavily on the Imagenet network architecture. At the bottleneck layer, rather than feeding the data into a dense layer, we used convolutions as described in Eq. 3.19, to upsample so that our output is the same size as our input. Using the mean squared error, Eq. 3.14, as the loss function, and using the apple photos as the input and output, we were able to learn a function to model the apples by running a stochastic gradient algorithm to minimize the loss. After running enough iterations so that the loss function begins to converge, the model can be deployed to run predictions. In Fig. 3.1, we see the result of passing various images of apples through the network. Even though the output image is not a perfect reconstruction, we see that the result still looks very similar to an apple. However, we also observe that the orange fruit reconstructions do not look like oranges, in fact they are quite apple-like. Because the network is trained only to reconstruct apples, the oranges have features that

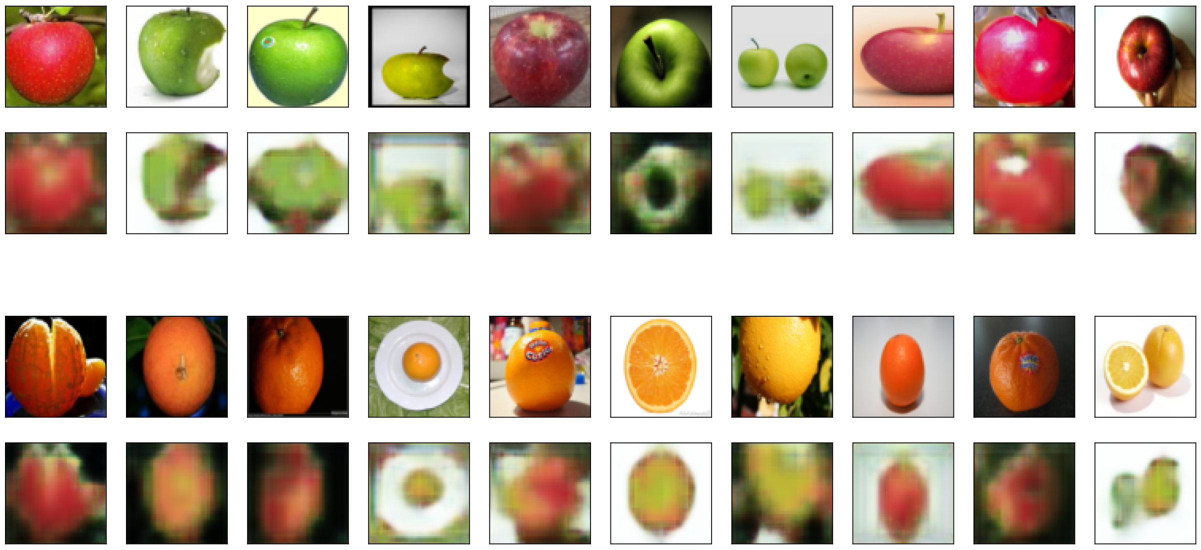
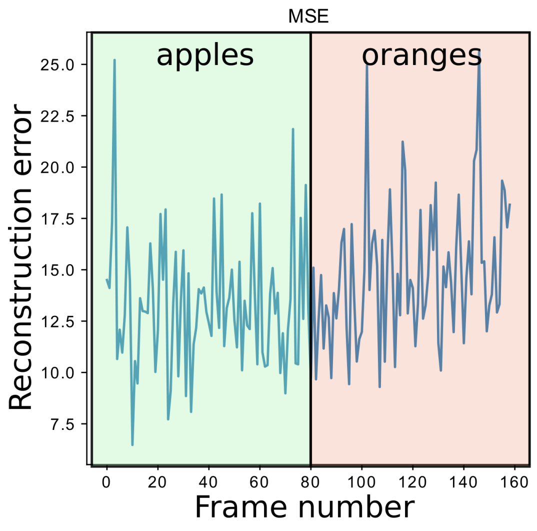


Figure 3.1: Reconstruction of apples and oranges. The top image is the original picture, the bottom is the reconstruction passed through the autoencoder. Best viewed in color.

are apple-like. The above result is confirmed in Fig. 3.2, where we see that on average the standard Euclidean distance from Eq. 3.14 between the reconstructions for the apples is smaller than that for the oranges.

We conclude that we can use autoencoders to learn a meaningful representation of what “apple” means. By taking advantage of this, we can use autoencoders to differentiate between apples and a class it has never seen; “oranges”. Because the class is balanced we generate an ROC curve [9], that compares the different thresholds and their associated false positive and true positive rates. These curves tell how accurately the classifier can predict outliers and how many false alarms it will generate for a threshold. By using the reconstruction error as a metric we can see how well it is able to discriminate between apples and oranges in Fig. 3.2, right hand side.

If we were to modify our network by substituting the bottleneck based on Eq. 3.17, we can train a new network keeping everything else relatively constant.



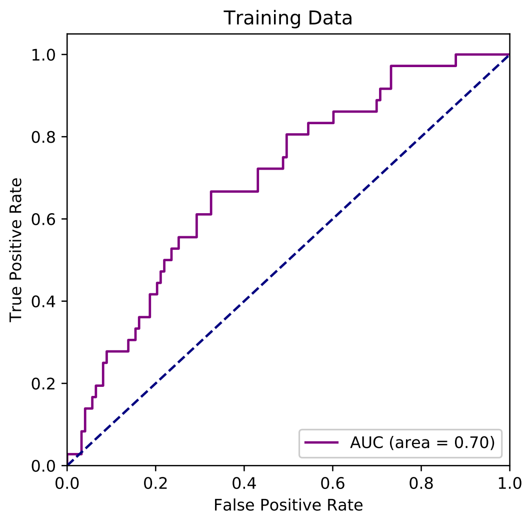


Figure 3.2: The graph on the top shows the mean squared error values of apples and oranges. The first 80 values correspond to the reconstruction error for apples and values $80 - 160$ correspond to the reconstruction error for the oranges. On the bottom we show the associated ROC curve.

The primary benefit of this is that now the feature space of the data is dispersed according to a multidimensional Gaussian (see Fig. 3.3) where the points represent apples in two dimensions of the feature space. If we sample from a random multidimensional Gaussian with the same shape as the bottleneck layer and pass those values through the decoding stage we can see what the network

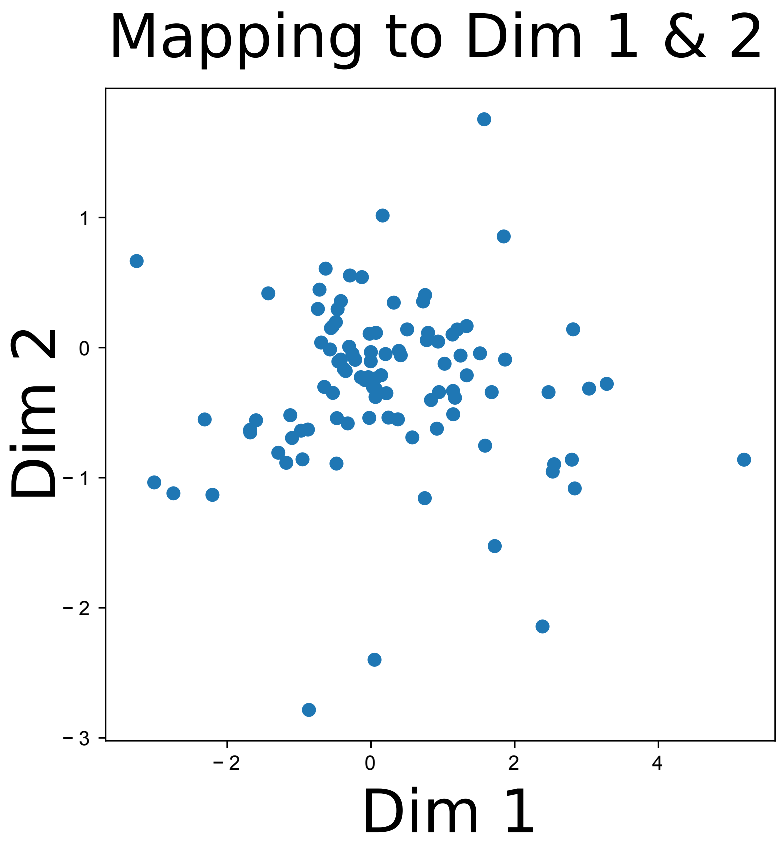


Figure 3.3: Dispersement of values for Variational autoencoder in the bottleneck layer. The dots represent the mapping of various apple images. We observe that the latent mapping looks like a Gaussian distribution.

thinks an “apple” looks like in Figure 3.4.

While the rest of the thesis does not deal with apples, this example illustrates the fundamental idea of how autoencoders can be used to detect anomalies as well as their benefit. Even though classifying between apples and oranges is a relatively easy task (even for a computer), a large benefit of this approach is that the network can “learn” to approximate what it means to be an “apple”. An additional benefit is that the network is not only capable of discriminating between apples and oranges but also between apples and other different images. In our work, “apples” represent the idealized data of what we expect our trackers to operate under normal conditions. The “oranges” are the

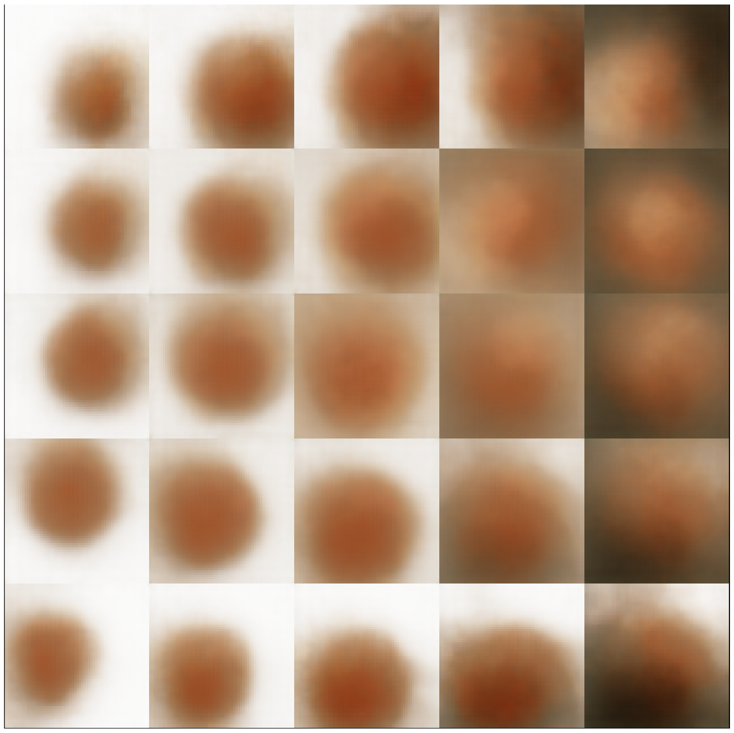


Figure 3.4: Variational autoencoder generated “apples”. Best viewed in color.

outliers that we hope to detect and penalize.

CHAPTER 4

DATA GENERATION AND AUTOENCODER DESIGN

In this chapter, we first explore HABDF, the tracker ensemble that this work builds upon. We look at the ensemble's performance and evaluate it using the OTB-50 benchmark. We briefly describe some of the issues with HABDF and what motivated our exploration of the alternatives to the Mahalanobis Distance weighing. Next, we explain how this ensemble can be used to generate training data. Here we also explain how this data can be transformed into datasets that allow us the test the anomaly detection performance of different algorithms. Finally, we explore and compare different methods, arriving at our proposed method of using Autoencoders.

4.1 HABDF Evaluation

First, we evaluate the reference method (HABDF) to determine its strengths and weaknesses. To do this, we take advantage of publicly available benchmarks.

4.1.1 Visual Tracking Benchmarks

To measure that the output of our data fusion method is working better than the trackers that comprise it, it is necessary to test the results on a benchmark. The OTB-50 benchmark is one of the most common tools used to evaluate various performance scores. Originally introduced in [85], the OTB-50 benchmark has 50 specific data sequences that it uses to provide different measurements of performance on various attributes. The most general of these measurements is the success, which measures how well the tracker can track the object throughout all of the image sequences. The OTB benchmark makes it possible to quantitatively evaluate the results generated by the tracker. Other publicly available visual tracking benchmark datasets include VOT [41] and ALOV [75]. We chose OTB-50 due to its simple integration and popularity. We would also like to note here that OTB-50 is part of OTB-100. OTB-50 selects 50 of the sequences from OTB-100, and changes these sequences periodically. This is so that there exists a smaller subset for faster testing.

4.1.2 HABDF OTB-50 Results

Our initial contribution was to evaluate the method proposed in [18] on a visual tracking benchmark. To do so, first we added the changes described in Section 3.1.1. The block diagram of our implementation is show in Fig. 4.1. We initially carried out separate evaluations of the four trackers that comprise our implementation of the proposed framework on the OTB-50 dataset. For STRUCK[1] and TLD[2] our results were 3% worse than the results reported in [35, 25], this is likely due to the changing sequences in the OTB-50 evaluation. For CMT and GOTURN, to the best of our knowledge, OTB-50 results are not publicly available so those had to be generated[3]. We then evaluated our approach on the same dataset with these same four trackers as part of our ensemble. We adjusted the values of ω proportionally to the success rate of each of the trackers on the OTB-50 dataset. The method showed a 5.5% increase in success relative to the best tracker in the ensemble and a 2.6% increase in precision. Since our method focuses on improving the overall robustness of the trackers, we expected the larger increase in success rate. The improvement in precision demonstrates that this method is not penalized by "imprecise" trackers such as CMT.

[1]the results were obtained using source code available at https://github.com/samhare/struck

[2]the results were obtained using source code available at https://github.com/klahaag/CFtld

[3]the results for CMT and GOTURN [27] were obtained using source code available at https://github.com/gnebehay/CMT and https://github.com/davheld/GOTURN respectively. We applied these methods "as shipped".

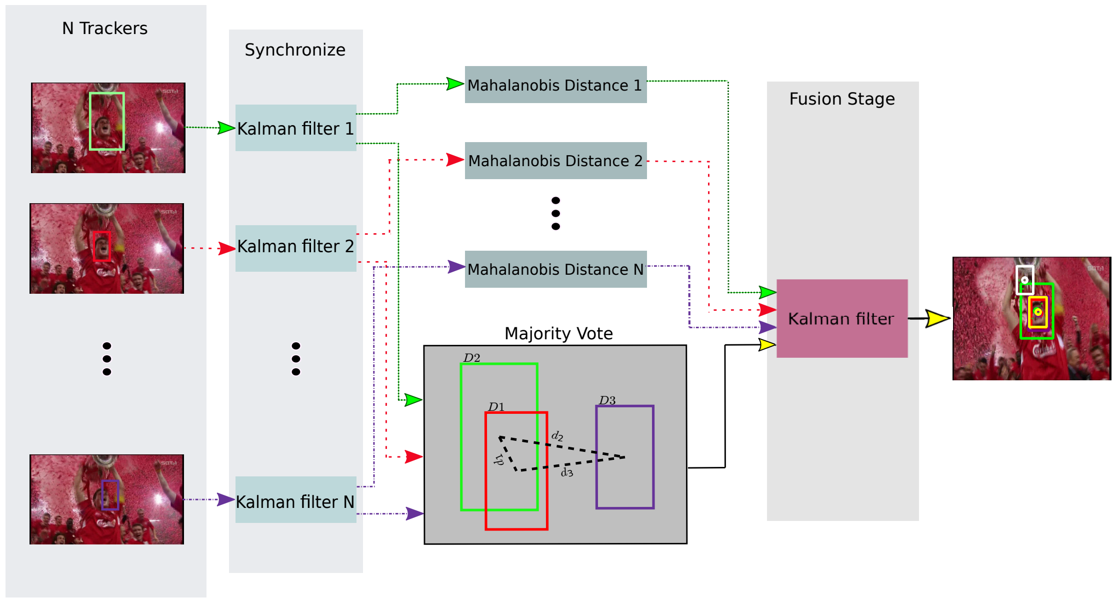


Figure 4.1: Schematic representation of the implementation of the baseline framework. Best viewed in color.

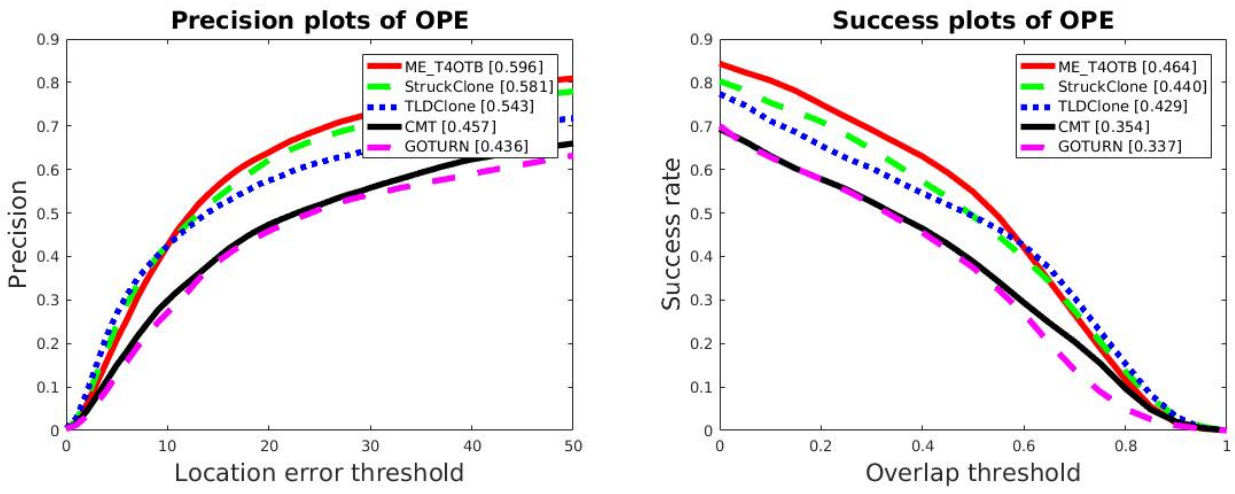


Figure 4.2: Results of our Tracker HABDF (referred to as ME_T4) on OPE for OTB-50

Our results in Figure 4.2 illustrate the performance of the proposed approach.We see that our method shows improvement in both precision and success. In particular, we see that our method has higher for location error threshold greater than 20. For success, we that our method has the highest success rate for overlap thresholds less than .6, this implies that our algorithm has more frames where this some overlap with the ground truth. Our ensemble leverages the individual strengths of each tracker to obtain higher levels of robustness throughout the various datasets. Even the best tracker in our ensemble performed poorly in certain scenarios, and despite providing the largest influence on the input, the other trackers helped improve performance overall.

Our ensemble is robust to failures from the lower ranked trackers such as GOTURN or CMT, and the failures of these trackers did not affect the overall performance when they occurred individually as seen in Figures 4.3 and 4.4. In the Figures our tracker is denoted by the yellow bounding box, and the color scheme for the other trackers is red/blue for TLD (blue when it is lost since TLD can make that determination), purple for GOTURN, green for CMT and white for Struck.

Figures 4.5 and 4.6 illustrate that our tracker is also robust to failures generated by the stronger trackers in the ensemble such as TLD or Struck.

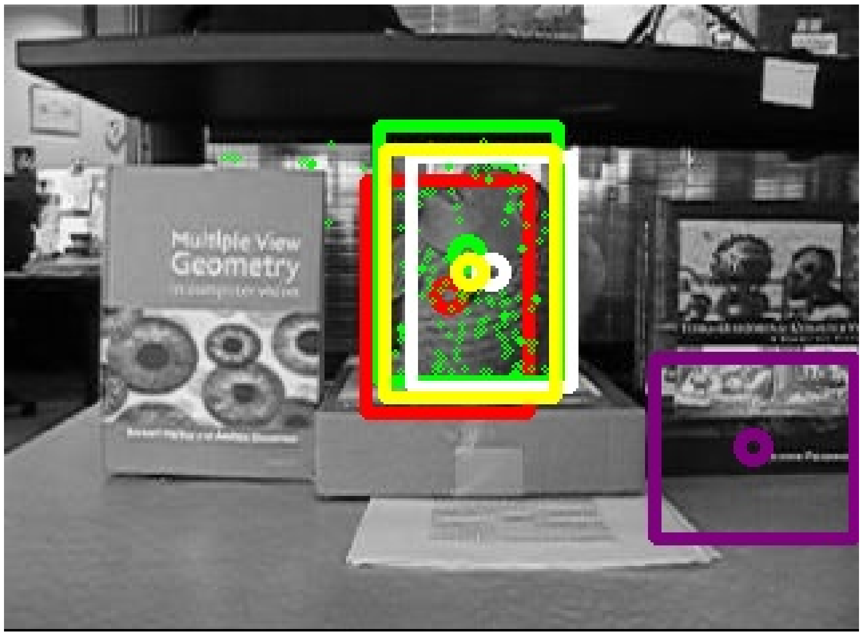


Figure 4.3: Robustness to failure from GOTURN. The red, green, white and purple boxes correspond to the outputs of TLD, CMT, STRUCK and GOTURN respectively. The yellow box is the output of the fused approach. Best viewed in color.

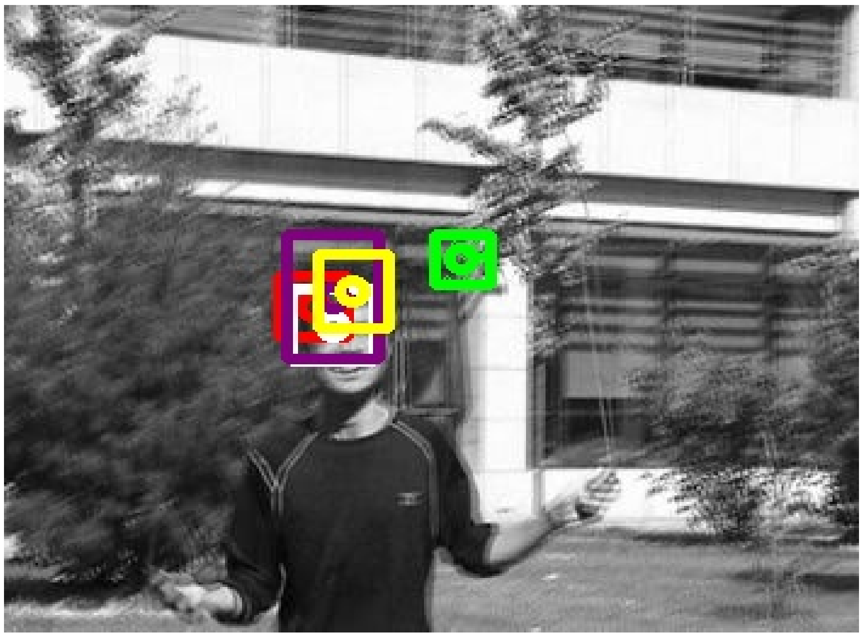


Figure 4.4: Robustness to failure due to CMT. See the caption of Fig. 4.3 for a description of the elements of the figure. Best viewed in color.

A simple majority voting approach would have allowed poorly performing trackers to degrade the overall results. Our method mitigates this issue by assigning weights based on the Mahalanobis distances of the measurements generated by each tracker, and also by incorporating previous knowledge about the performance of the individual trackers. In Figures 4.3 and 4.4, it can be observed that these anomalous measurements have a minimal effect on the overall tracking result. This is seen by the yellow fused result, that chooses to follow the other trackers rather than the anomalous one. In the first subfigure, GOTURN's distance from the other trackers assigns the tracker a high weight due to Eq. 3.8. Hence, GOTURN has a minimal effect on the final estimate and continues to do so due to the motion model associated with the resultant tracker.

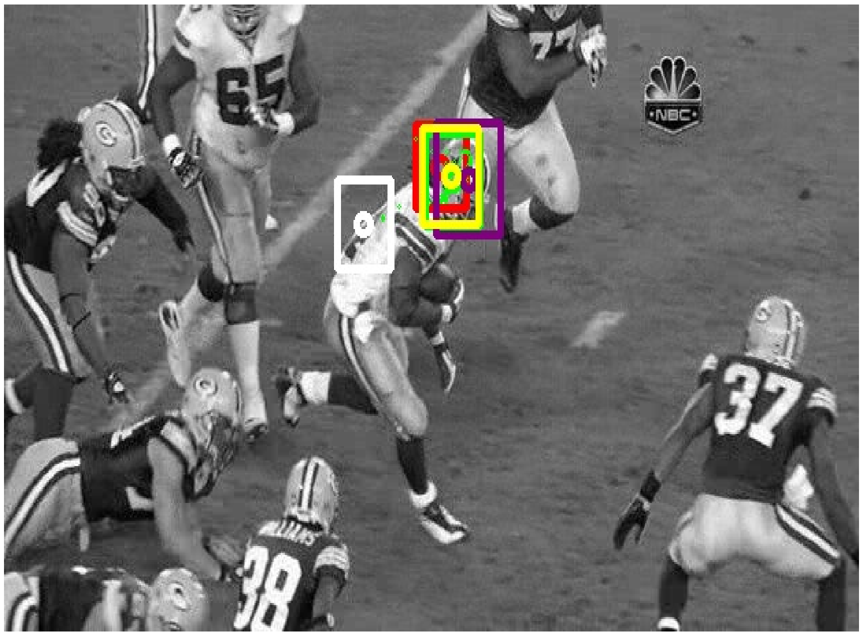


Figure 4.5: Robustness to failure from the strongest tracker in the ensemble. See the caption of Fig. 4.3 for a description of the elements of the figure. Best viewed in color.

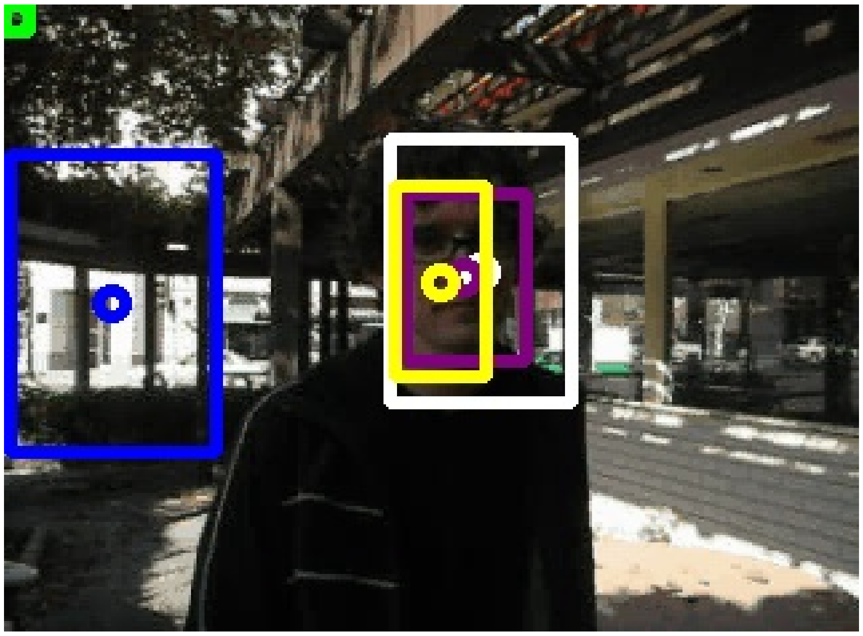


Figure 4.6: Robustness to the failure of multiple trackers. See the caption of Fig. 4.3 for a description of the elements of the figure. Best viewed in color.

In Figure 4.5, the best tracker in the ensemble, STRUCK, has failed. Because our weighing mechanism is not just a weighted voting scheme, our tracker is able to disregard the measurements from Struck. In the second figure, we see that two trackers are lost, but our ensemble is still able to perform very well. By taking advantage of the proximity between Struck and GOTURN as dictated by Eq. 3.8 these trackers have a much higher influence on the output. CMT and TLD, on the other hand, are far from any other tracker, accrue a higher penalty and do not significantly influence the output. It is also important to note that because of the weights applied using the Mahalanobis distance, the fusion approach penalizes erratic performance from trackers. The weights generated by the Mahalanobis distance allow the framework to smooth out the estimate and engender a more steady and more robust output. Besides positively impacting success and precision, the method also significantly increased the score where all the trackers had a similar score for the specific metric. This benefit is especially obvious for the OPE of out-of-plane rotations. Despite the significant improvement in most scenarios, in the rare situations where performance was drastically different among trackers, a decrease in performance was observed relative to the best tracker in the ensemble. When multiple trackers are significantly worse than the best trackers, the performance may decrease. This is especially obvious for the case of low resolution images in which GOTURN and CMT perform very poorly and hence degrade the overall performance.

We present the complete results of our tracker relative to demonstrate that the fusion technique clearly increases robustness. Unfortunately, this increase in robustness means that sometimes the individual strengths of a tracker is lost. In particular, we point the motion blur and low resolution scenarios in Table 4.1.

As Table 4.1 indicates, our approach improves the performance in 8 of the 12 scenarios including; illumination, out-of-plane rotation, scale variation, occlusion, deformation, in-plane rotation and background clutter. In the cases where the performance decreases, it is important to note the large discrepancy between the best tracker and the other trackers in the ensemble. Because the method uses the confidence generated by the Kalman filter, when multiple

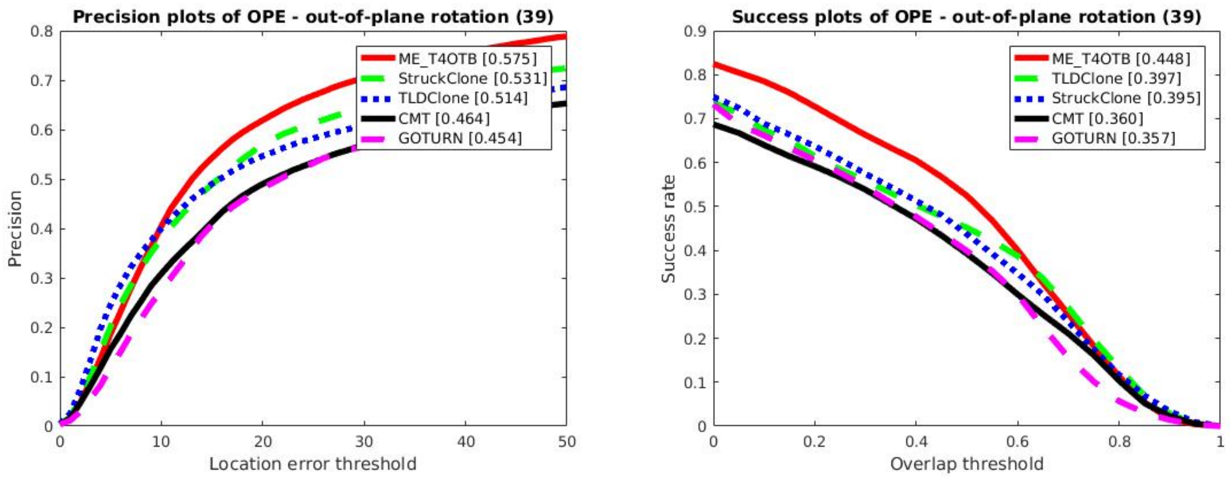


Figure 4.7: The increase in performance for out-of-plane rotation.

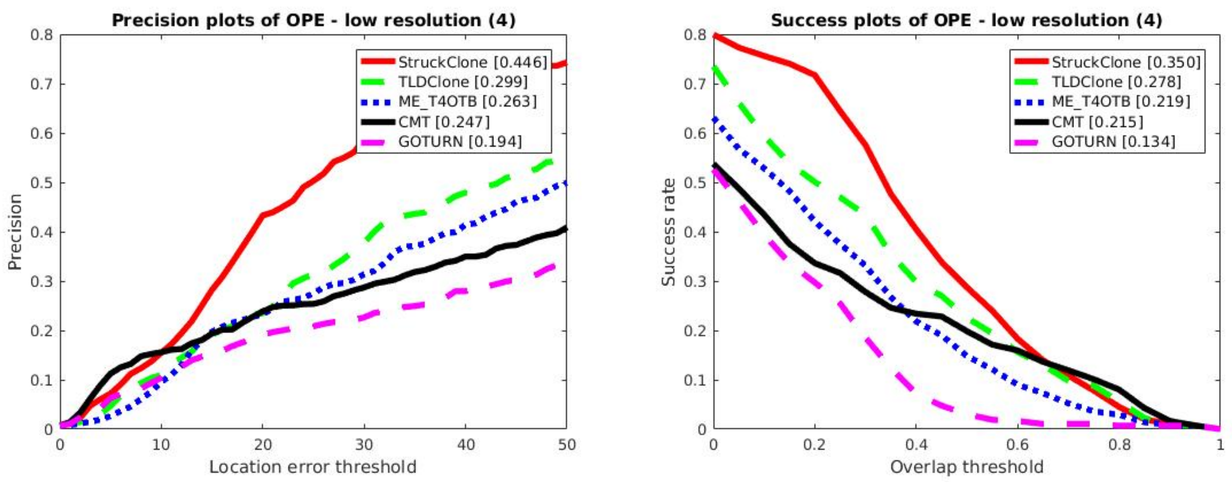


Figure 4.8: The decrease in performance for low resolution.

trackers show poor performance, our results can be negatively affected. The improvement is most obvious and prominent when the trackers show similar performances. This problem can be mitigated by either refraining from using trackers that perform very poorly under certain scenarios or by adjusting its prior weight according to its worst-case performance.

4.1.3 HABDF Issues

A Bayesian data fusion approach was applied to the problem of vision-based target tracking and showed promising results in the OTB-50 dataset. Significant increases in robustness were observed despite the weaknesses of certain trackers. The method provides an adaptive framework that uses both the local statistics generated by each tracker as well as a weighted majority voting mechanism to determine the target bounding box at each frame. Pretraining is not required, and the method is robust in practical scenarios due to its ability to integrate multiple sources of information.

One simple way to extend this work would be to consider the problem of outlier detection. If it is known with high probability that a tracker is lost, it can

Table 4.1: Summary of results on OPE

Scenario	Best Tracker	Best Tracker Score (Precision /Success)	Worst Tracker	Worst Tracker Score (Precision /Success)	Fusion Score (Precision /Success)	Percent Change Relative to Best Tracker
Total	Struck	.581/.440	GOTURN	.436/.337	.596/.464	+2.581% / +5.454%
illumination	Struck	.581/.413	GOTURN	.347/.291	.524/.427	+1.158% / +3.389%
out-of -plane rotation	Struck /TLD	.531/.397	GOTURN	.454/.357	.575/.448	+8.286% / +12.846%
scale variation	Struck	.562/.386	CMT	.435/.327	.574/.432	+2.135% / +11.917%
occlusion	Struck/TLD	.521/.408	CMT /GOTURN	.404/.311	.548/.431	+5.182% / +5.637%
deformation	Struck	.516/.414	CMT	.373/.301	.568/.450	+10.078% / +8.696%
motion blur	Struck	.487/.406	GOTURN	.300/.254	.450/.366	-8.222% / -10.929%
fast motion	Struck	.520/.424	GOTURN	.410/.282	.455/.377	-14.286% / -12.467%
in-plane rotation	TLD	.552/.435	GOTURN	.332/.326	.556/.439	+0.725% / +0.920%
out of view	Struck	.482/.444	GOTURN	.332/.316	.465/.441	-3.656% / -0.680%
background clutter	Struck	.530/.429	CMT	.341/.263	.547/.437	+3.207% / +1.864%
low resolution	Struck	.446/.350	GOTURN	.194/.134	.263/.219	-69.582% / -59.818%

be reinitialized. Fault detection and correction would improve overall success and greatly assist in generating a better, more robust framework [83]. In particular, it would alleviate the issues that occur when some trackers perform substantially worse than the others. One avenue is to simply use the confidence generated by the Mahalanobis distance to determine if a tracker is an outlier. Possible alternative approaches include supervised ideas such as those presented in [87]. Keeping with the Bayesian and unsupervised nature of the proposed framework, an unsupervised approach is more fitting and some possible ideas include those presented in [57, 59, 52].

4.2 Data Acquisition

In order to address the issues with the Mahalanobis distance in HABDF, such as a lack of long-term dependency, we propose using a machine learning approach. Our goal is to use data from HABDF and the constituent trackers as training data. By using machine learning, we can teach our algorithm to statistically discriminate poor data.

In this section, we explain how we use HABDF as a tool to acquire data from multiple trackers. Afterwards, we explain how this data is partitioned and used to train our network. The OTB-100 benchmark [84] is one of the most common tools used to evaluate various performance scores of visual tracking algorithms. It contains 100 video sequences that it uses to measure tracking accuracy and robustness. These measurements then allow tracking algorithms to be assigned a score and compared to other trackers based on these two criteria. This is also beneficial for our purposes because a common evaluation tool for a tracker is to use half of OTB-100 in OTB-50. This is beneficial to us, because we will effectively have two separate datasets.

Our goal is to separate "normal" and "anomalous" data using only the results available from the tracker in an offline manner. In our work, the set of all frames is $\mathcal{F} = \mathcal{F}_S \cup \mathcal{F}_O$, where $\mathcal{F}_S$ refers to all the normal frames and $\mathcal{F}_O$ refers to anomalous frames of different types. Section 4.2.2 describes how we determine which frames belong to each category. We use 51 data sequences from OTB which we refer to as $\mathcal{F}^{(1)}$ to train our algorithms. The other 49 sequences $\mathcal{F}^{(2)}$ are used as a test set. At each , the Kalman filters in Eqs. 3.1 and 3.2 produce N state estimates $x_m \in \mathbb{R}^D$, where D is the dimension of the target state and N is the number of trackers used in the ensemble. Since in our application, $D = 8$ and

$N = 4$, we obtain a 32-D vector at every frame.

Additionally, we acquire data from a third source [47]. This additional benchmark introduces 78 unique sequences in addition to having 50 sequences in common with OTB. We run the tracker on this sequence as well to generate an auxiliary training set $\mathcal{F}^{(aux)}$.

4.2.1 Proposed Approach

We propose a method for tracker data anomaly detection that can act as a generic framework and allows for modularity in its implementation. Our data anomaly detection framework can be divided into 4 separate sections: 1) acquire training and testing data; 2) define a deep neural network architecture; 3) fine tune our network; 4) test our network on how well it can differentiate between normal operation and the anomalies associated with various trackers.

4.2.2 Data Partitioning

Let $f_m \in \mathbb{R}^{N \cdot D}$ represent the feature vector corresponding to the concatenation of the outputs of all the Kalman filters that is equal to,

$$f_m = \left[x_m^{(1)}, x_m^{(2)}, \ldots, x_m^{(N)}\right]. \tag{4.1}$$

Each $x_m^{(n)}$ corresponds to the state vector of one of the N trackers as described in Section 3.1. Figure 4.1 details how the framework is implemented. Let $b_m^{(n)} = [x, y, h, w]$ be the bounding box generated by the n-th tracker at frame f_m. Let $\mathcal{F}$ be the set of all frames in all the video sequences. To split the data, the Jaccard index is used to determine whether the data is an inlier or outlier. For each result frame, the Jaccard index is computed as

$$J(b_m) = \frac{|b_m \cap \bar{b}_m|}{|b_m \cup \bar{b}_m|}, \tag{4.2}$$

Where $\bar{b}_m$ is the ground truth for that frame. By calculating the Jaccard index for every tracker at every point we can divide our total data into $N + 1$ distinct subsets.

1. When all trackers at a frame have a Jaccard index greater than τ we consider this "normal" data,

$$\mathcal{F}_S = \left\{ f_m \in \mathcal{F} | J(b_m^{(n)}) > \tau \right\}. \tag{4.3}$$

   for $n = 1, ..., N$ and $m = 1, ..., M$.

2. When all but one tracker at a frame have a Jaccard index greater than τ we consider this "locally anomalous" data for that specific tracker. This creates N different datasets (one for each tracker)

$$\mathcal{F}_{O^{(k)}} = \left\{ f_m \in \mathcal{F} | J(b_m^{(k)}) < \tau \wedge J(b_m^{(n)}) > \tau \right\}, \tag{4.4}$$

   for $n = 1, ..., N, n \neq k$, for $m = 1, ..., M$.

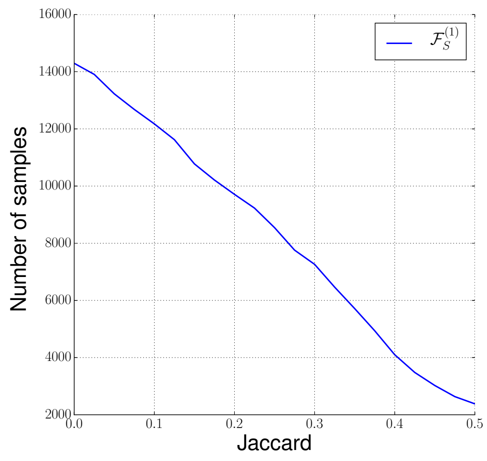


Figure 4.9: Relationship between the number of normal samples as a function of the Jaccard index. Higher Jaccard indexes present the additional challenge that a smaller percentage of data can be used for training.

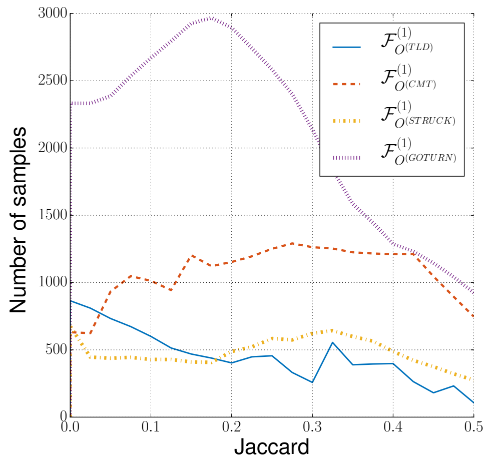


Figure 4.10: Relationship between the number of anomalous samples as a function of the Jaccard index.

This partition is done for all partitions of $\mathcal{F}$ including $\mathcal{F}^{(aux)}$, $\mathcal{F}^{(1)}$ and $\mathcal{F}^{(2)}$. In total, this generates $3(N+1)$ partitions, although not all are used.

4.2.3 Mahalanobis Distance Baseline

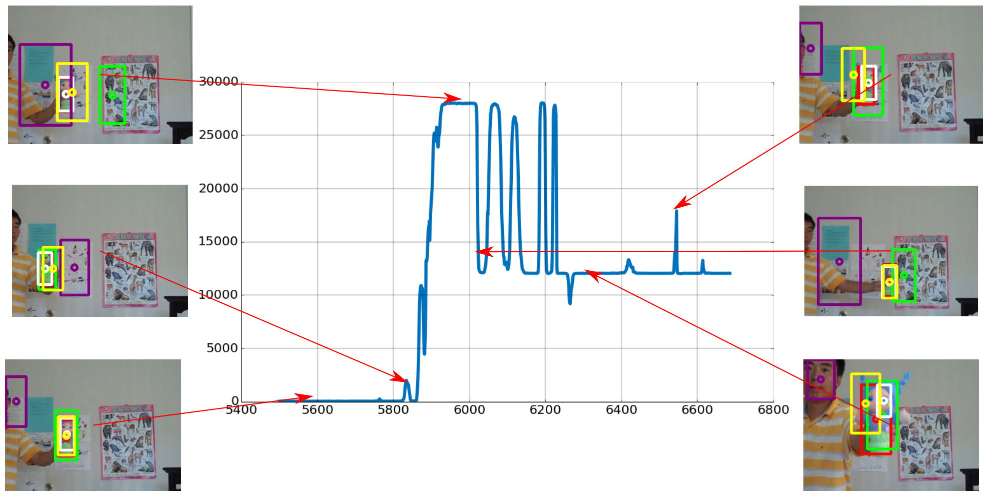


Figure 4.11: Illustration of the results generated using the approach based on Eq. 4.5 for the sequence *Doll*. The red, green, white and purple boxes correspond to the outputs of TLD, CMT, STRUCK and GOTURN respectively. The yellow box is the output of the fused approach. This method is capable of detecting outliers but struggles in complex scenarios where motion is highly non-linear and the Kalman filters covariance fails to capture that appropriately, as indicated by the frames in which there are lost trackers but the value of W_Γ shown in the center graph is relatively low.

After running our trackers on the OTB-100 dataset, we generate 29,492 total frames for $\mathcal{F}^{(1)}$ and 29,550 for $\mathcal{F}^{(2)}$ using the methods described in Sections 3.1 and 4.2. By applying the data partitioning method illustrated in section 4.2.2 for various values of τ we generate diverse sized datasets. We first observe how well our method can differentiate between $F_{O^{(k)}}$ and F_S as this was the easier problem.

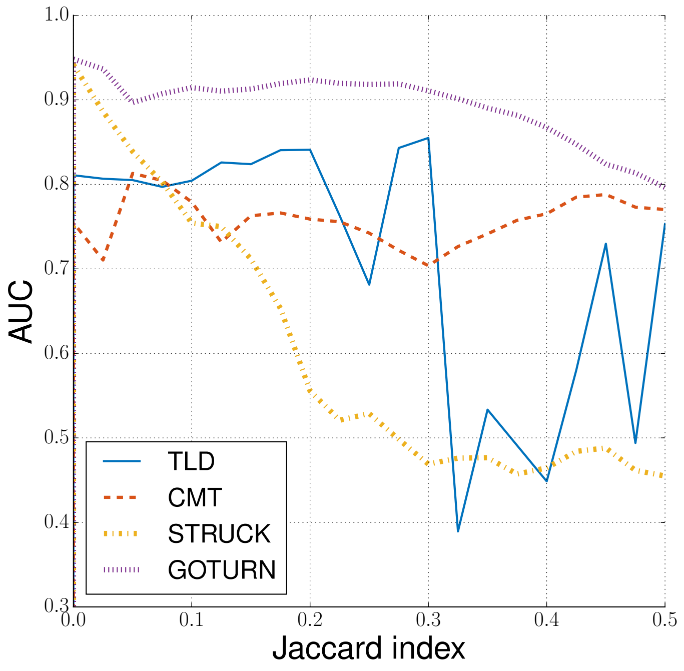


Figure 4.12: By using the proposed Mahalanobis distance method in [68], we generate the area under the curve (AUC) for various values of τ. We see that this method particularly struggles with STRUCK and higher τ.

One important consequence of our method is that for higher values of τ, our $\mathcal{F}_S$ set becomes smaller. This implies that for more stringent values of τ we have less training data, which is supported by Figs. 4.9 and 4.10 which show the number of samples of $\mathcal{F}_S$ and $\mathcal{F}_{O^{(k)}}$ for various τ values. However, $\mathcal{F}_{O^{(k)}}$ does not necessarily follow this rule due to the increasing τ which brings some of previously "normal" data $\mathcal{F}_S$ into the "local outliers" $\mathcal{F}_{O^{(k)}}$ subset as indicated in Fig. 4.10. To evaluate the success rate of our methods, we apply the approach in Section 4.2.1 in order to determine whether we can separate frames in $\mathcal{F}_S^{(2)}$ and $\mathcal{F}_{O^{(k)}}^{(2)}$ for different values of τ.

4.2.4 Baseline Approach

In HABDF, each tracker n generates a value $w_{\Omega(n)}$, it is then possible to sum up all these values and use that to act as threshold to determine whether the

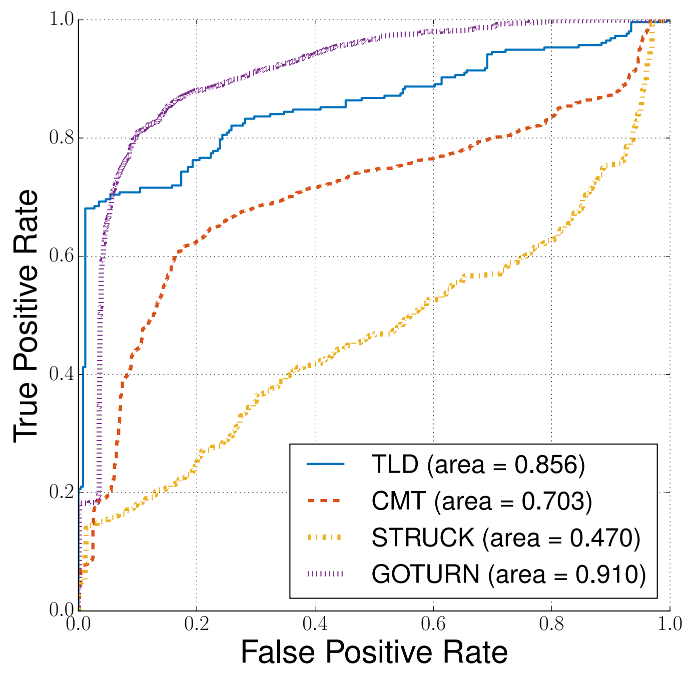


Figure 4.13: ROC curves for the Mahalanobis distance method at $\tau = .3$ on a test set. We notice that the Mahalanobis distance particularly struggles with STRUCK and performance is not consistent across all trackers.

current frame is "normal" $\mathcal{F}_S^{(2)}$ or an "anomaly" $\mathcal{F}_{O^{(k)}}^{(2)}$ according to

$$W_\Omega = \sum_{i=0}^{N} w_{M(n)}, \tag{4.5}$$

Fig. 4.11 illustrates the values of W_Ω for several frames of one illustrative video sequence. We present the results in Fig. 4.13 for a τ of .3 and show the area under the ROC curve for τ values between 0 and .5 in Fig. 4.12.

4.2.5 Supervised Approaches

We examined the performance of two common approaches for dataset classification: Support Vector Machines (SVM) and K Nearest Neighbors (KNN). We compared these methods by training using $\mathcal{F}_S^{(1)}$ and $\mathcal{F}_{O^{(k)}}^{(1)}$ as our two separate classes for all the trackers.

All three performed well on our training dataset $\mathcal{F}_S^{(1)}$ and $\mathcal{F}_{O^{(k)}}^{(1)}$. However, when the weights were saved and applied to $\mathcal{F}_S^{(2)}$ and $\mathcal{F}_{O^{(2)}}^{(2)}$ we observe that these methods fail to capture the non-linear nature of this problem as seen in Fig. 4.14 and Fig. 4.15. In this problem, it can be inferred that a semi-supervised or unsupervised method is necessary to capture the complexity inherent in our data.

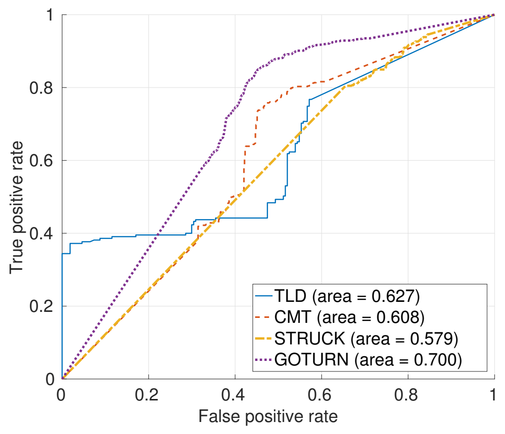


Figure 4.14: Outlier detection using a KNN classifier with 10 neighbors on testing set. $\tau = .3$.

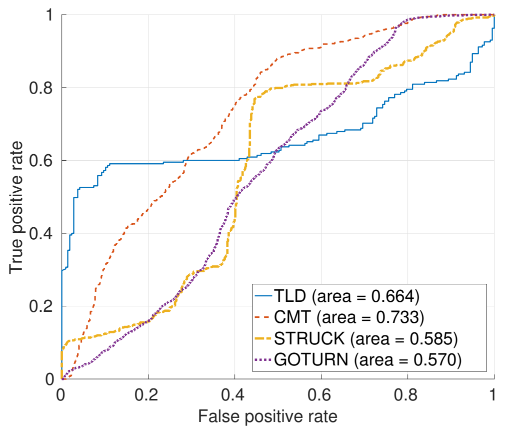


Figure 4.15: Outlier detection using a SVM classifier on testing set. $\tau = .3$.

The issue with supervised approaches is that we must acquire a large sample of data of both the positive and negative class. Not only are supervised approaches incredibly data dependent, they also require that the data for both classes is entirely representative of the space. If we were trying to build a classifier that determines whether an image is a fruit we would not only need examples of fruits but also of examples of everything that is not a fruit including; cars, avocados, Dali paintings, etc. An unsupervised approach such as k-means sounds appealing but is problematic because there is no guarantee that the resulting partitions will be the outliers and inliers. This problem naturally motivates the desire for a semi-supervised approach where we can feed examples of what we expect and hope the algorithm can discriminate between that and things it is not used to. In our example, we fed examples of fruits to our approach and the algorithm would learn a latent representation of “fruit”. This allows outliers and anomalies to be interpreted as data points that are farther away on the manifold to the expected class.

4.3 Deep Autoencoder

We present our proposed approach of using the autoencoder as described in Section 3.2. Our approach was to first acquire the output of our trackers from their local Kalman filters, concatenate them horizontally and use that as the input data into our network. The concatenation process is described in Eq. 4.1, while the basic construction process for the network is described in Section 3.2, we use the acquired data from our frames with an overlap ratio τ with the ground truth to generate our data of what we call “normal” operation. This normal data corresponds to $\mathcal{F}_S$, which was described in more detail in Section 4.2.2. Furthermore, to validate our network we train our model only with $\mathcal{F}_S^{(1)}$; which represents our “apples”. Similarly, $\mathcal{F}_{O^{(k)}}^{(1)}$ corresponds to the data of the different failing trackers. In order to demonstrate that our method is robust, we evaluate how well how our method can detect outliers from all of our trackers. This prevents scenarios where our method is better at predicting a generally more faulty tracker but struggles with predicting anomalies for a better tracker because the better has less data. This is done to ensure we pick the most robust and general algorithm.

To test our network, we use $\mathcal{F}_S^{(2)}$ and $\mathcal{F}_{O^{(k)}}^{(2)}$, which present the positive and negative classes of our data but from a different data sequence. We scale the data using a minimax algorithm in order to ensure that the data is approximately in the range that the autoencoder can generate with the output function being a hyperbolic tangent [39]. This scaling is done by finding the max and min value for every feature in $\mathcal{F}_S^{(1)}$. An additional reason is that since this is meant for real-time application the scaling has to use the same weights each time. We also note that is additionally beneficial in preventing overfitting. In our first attempt, we performed the minimax scaling on the datasets separately and generated very impressive results on the training set. Unfortunately, performing the prescaling on the datasets individually did not work on novel examples. This introduced a bias because the network could learn to differentiate the datasets based on how they were scaled. We present the scaling method below

$$f_m = \frac{2(f_m - f_{min})}{(f_{max} - f_{min})} - 1. \tag{4.6}$$

Here we first compare the performance of the standard Deep autoencoder with the results shown in Fig. 4.13 and Fig. 4.12. Our initial proposed autoencoder is shown in 4.16.

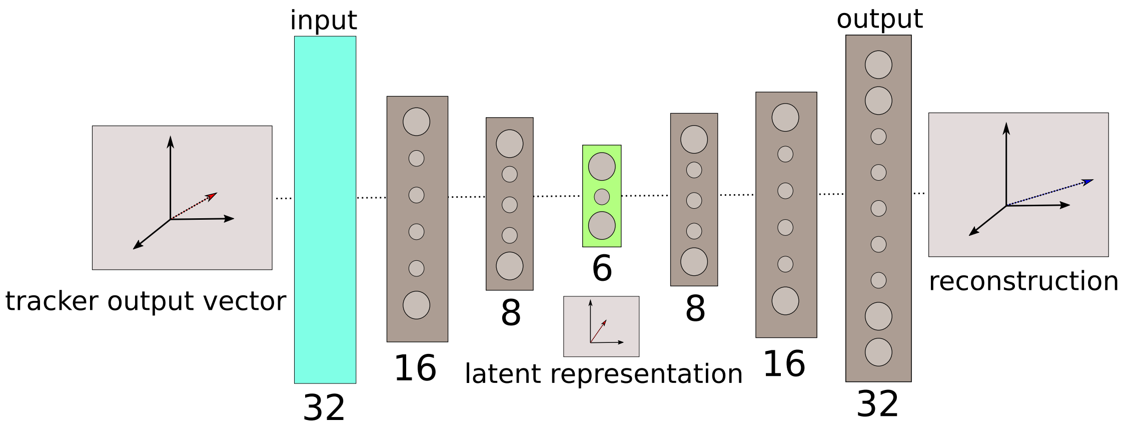


Figure 4.16: Schematic representation of the implementation of the autoencoder.

Initial results were promising but still left room for improvement. To compare our autoencoder to our original method, we look at performance where $\tau = .30$ and consider the AUC for a range of τ from 0 to .50.

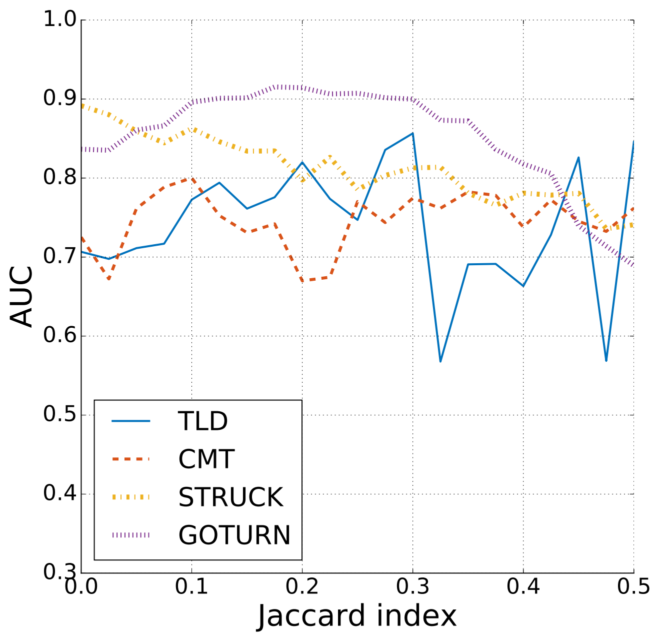


Figure 4.17: Area under the curve for the trackers for various values of τ using the method described in Section 3.2 and shown in Fig. 4.16. We note that similarly to the Mahalanobis distance, higher τ pose a tougher problem and performance is worse for small τ as well.

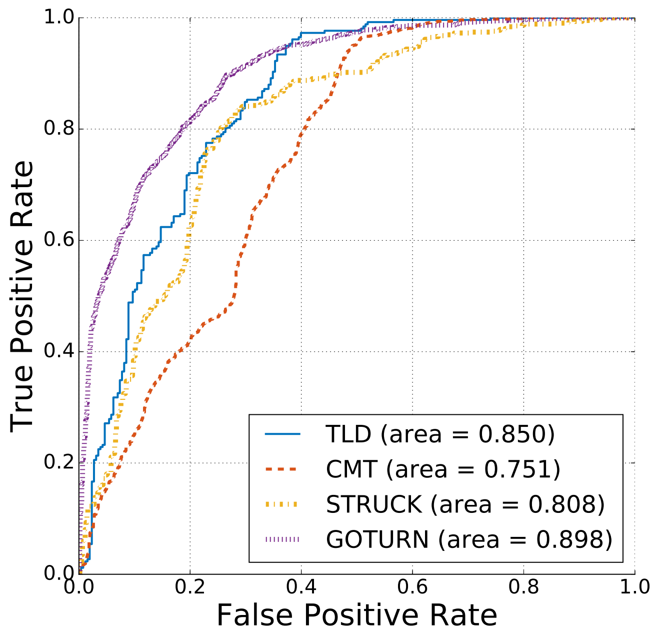


Figure 4.18: ROC curves for the autoencoder method at $\tau = .3$ on a test set.

While performance was great in the training set and much better in the test set with the autoencoder method than other supervised methods, we propose to improve performance of the autoencoder anomaly detection approach whose results are shown in Fig. 4.18. In particular, our performance at lower levels of τ, the autoencoder underperforms relative to the Mahalanobis baseline as seen in Fig. 4.17. To improve performance, we hypothesized that overfitting was an issue of concern. To address overfitting we selected two common techniques used to performance in autoencoders.

4.3.1 Tools to Improve Autoencoder Performance

To address the issues in the section above we proposed 2 common methods to address overfitting: Denoising [82] and L2 regularization [63] .

The first method is known as denoising and has been applied in many autoencoder applications including [56]. Gaussian noise is added during training to the input x, to generate the corrupted input $\acute{x}|x \sim N(x, \sigma^2 I)$. Rather than learning to reconstruct the input, it learns to reconstruct a corrupted input. This added detail assists the network in learning a better latent representation [55]. From a conceptional point of view this technique increases robustness by making sure the network is not learning a dictionary to encode the data.

Additionally, we propose L2 regularization to assist our network.

$$\Phi(\theta) = \sum_{x \in \Xi} \|x - \tilde{x}\|^2 + \varepsilon \sum \|W\|^2, \tag{4.7}$$

where ε corresponds to the regularization parameter. This is a technique that prevents network from overfitting by assuming a Gaussian prior. Practically this encourages the network to learn many small weights rather than having a few large weights [21].

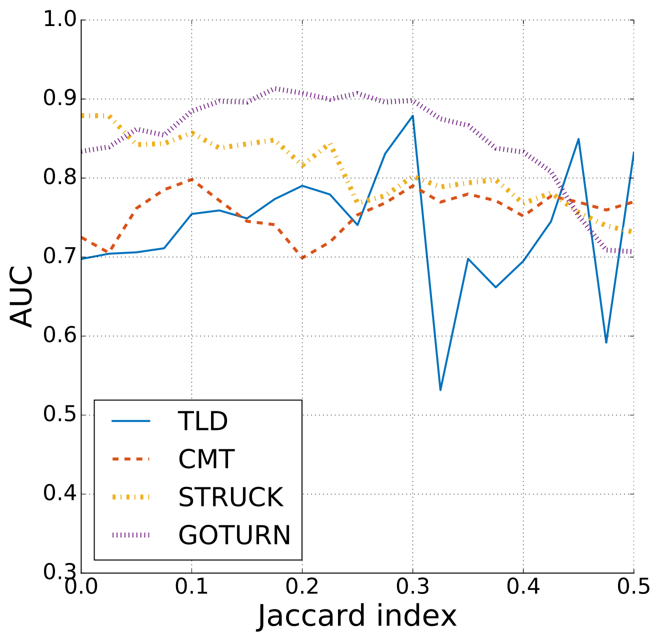


Figure 4.19: Area under the curve for the trackers for various values of τ using the method described in 3.2 and shown in Fig. 4.16. We note that similarly to the Mahalanobis distance, higher τ pose a tougher problem.

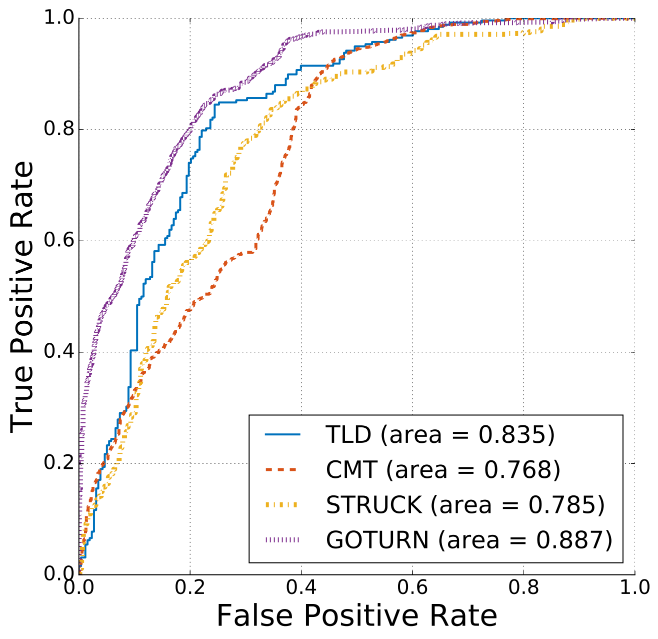


Figure 4.20: ROC curves for the Autoencoder method at $\tau = .3$ on a test set.

We see that these methods provide some marginal improvement in Fig. 4.19. In particular TLD, CMT and STRUCK seemed to improve for $\tau > .3$ as shown in Fig. 4.20. However, it seems that there could be some improvement possible, so, we explored further avenues.

4.3.2 Variational Autoencoder

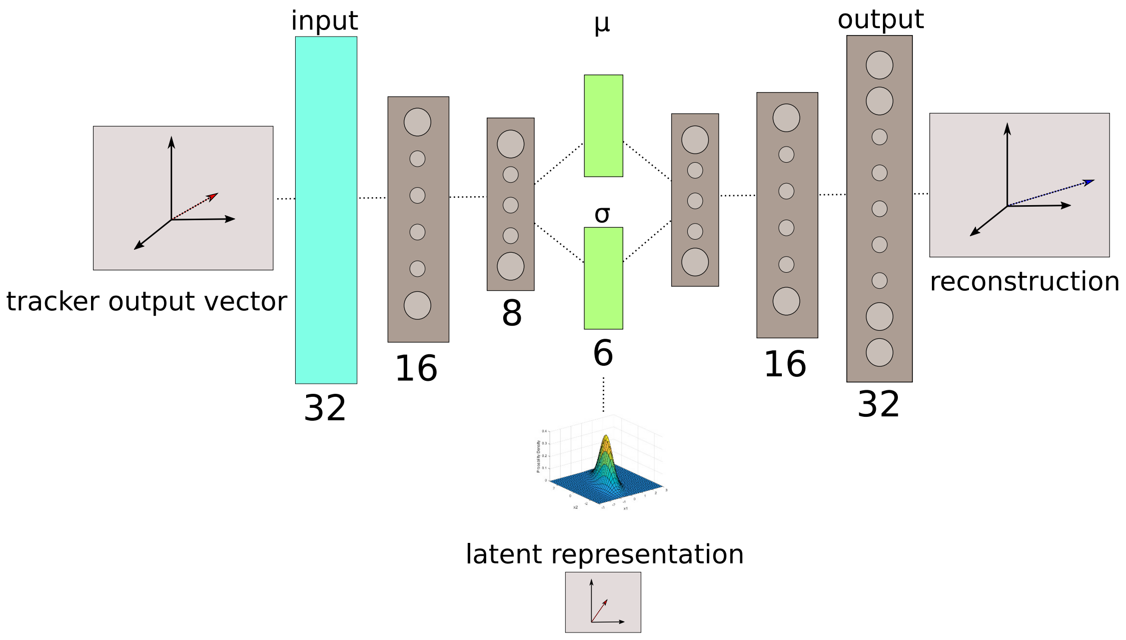


Figure 4.21: Schematic representation of the implementation of the variational autoencoder framework.

The variational autoencoder (VAE) was first described in 2013 in Kingma [37]. That approach further built upon the statistical nature of the autoencoder by modeling the latent layer as a Gaussian distribution. Rather than the latent layer being a complex function, the multidimensional Gaussian model learns a much richer and expressive representation of the data. Because we want to avoid overfitting while still learning a latent representation of our data, we explored the VAE as tool to learn the latent model for when our trackers are properly working. Our model is shown in Fig. 4.21.

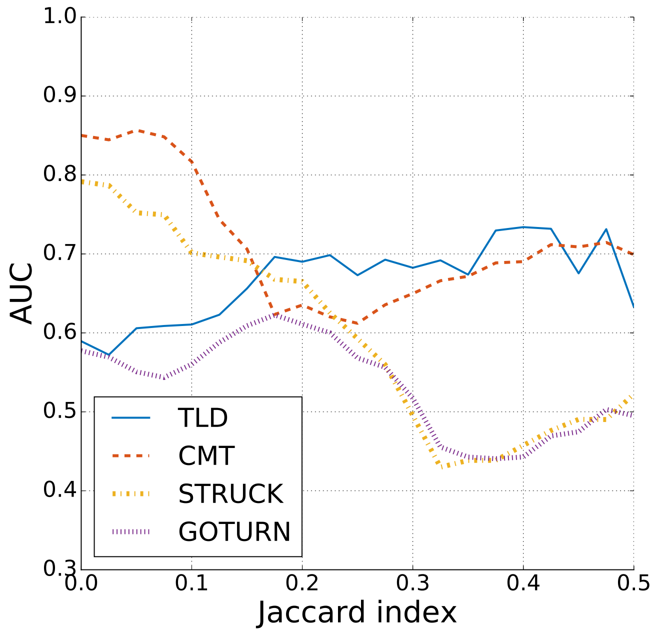


Figure 4.22: Area under the curve for the trackers for various values of τ on the test set using the method described in 3.2 and shown in 4.21. We note that the standard Autoencoder is a better discriminator.

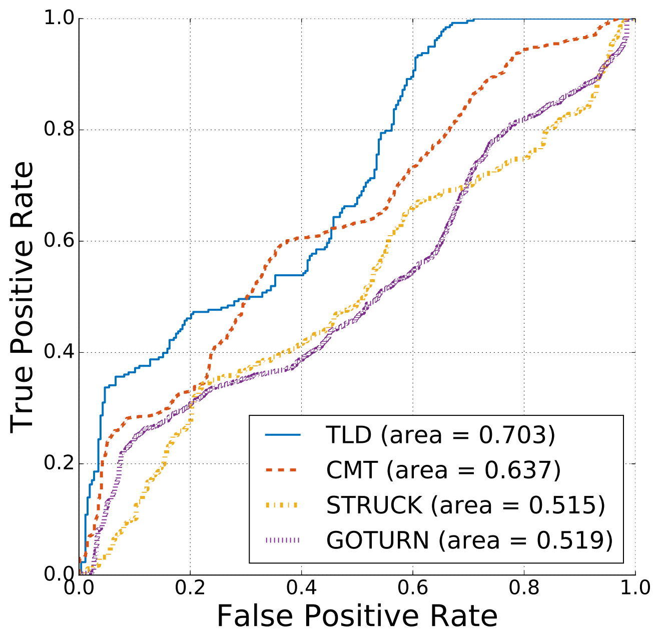


Figure 4.23: ROC curves for the Variational Autoencoder method at $\tau = .3$ on a test set.

We see in Fig. 4.22 Fig. and 4.23 that the variational autoencoder is poor at detecting anomalies in our scenario. Although it is documented as a strong modeling mechanism, the model's area under the curve for the all of the trackers is between .12 and .4 worse than the baseline autoencoder. With these results, we concluded that there was still some advantage that the Kalman filter possessed that our method did not have. The Kalman filter uses a Hidden Markov model that takes advantage of the previous frame. This means that the Kalman filters possess some time dependent information. This led us to explore approaches that used multiple frames as a source of information. We introduce a model that uses the previous two frames to predict the current frame. Our method remains an "autoencoder" because we keep our structure of using a latent layer to force the network to learn an abstract representation of expected data.

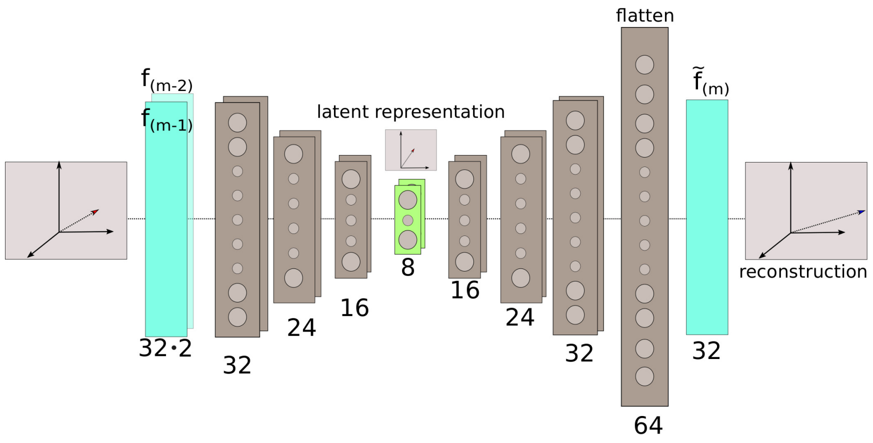


Figure 4.24: Our proposed network. The input vector consists of the feature vectors computed at two consecutive frames, $f_{(m-1)}$ and $f_{(m-2)}$. The dimensionality of each layer is shown below the layer. In particular, the bottleneck layer has dimensionality $L_3 = 8$.

4.3.3 Convolutional Autoencoder

Our approach is based on an autoencoder. As mentioned previously, autoencoders are neural networks that try to model the target function input = output. By performing backpropogation, the network learns a model for the latent space of the data which means that it learns to represent the points in a lower dimensionality space. Our network utilizes a 1D convolutional encoding deep architecture inspired in part by Krizhevsky's work in [42] where the autoencoder has a structure that progressively stacks smaller layers until a bottleneck layer, at which point every subsequent layer is larger until the last layer has the same size as the input. We modify this slightly by employing a flatten operation [42] in the penultimate layer. Figure 4.24 provides a visual representation of the architecture. For any input example f_m, we generate a hidden representation through a series of applications of the activation function $\alpha(f_m^i) \in \mathbb{R}^{L_l}$, where L_i is the dimensionality of the l-th hidden layer and $f_m^{(l-1)}$ is the output of the previous layer. Based on experimentation, we chose to use the hyperbolic tangent activation function. The dimensionality of the inner layers is represented in Fig. 4.24.

4.3.4 Training Details

Data is partitioned such that two previous frames are used to predict the current frame. The network is trained so that the value in Eq. 4.9 is minimized. Training is predicated upon using the current frame $\mathcal{F}(t)_S$ as the target and the two previous frames as inputs to the network. We scale the data to the interval $[-1, 1]$ and use the Adam optimizer [36] with a step size of 0.0001 as our optimizer with Eq. 4.9 as our cost function. The batch size as well as the number of epochs are both set to 50 through empirical testing. We utilize early stopping with a patience of 10, and perform a 33% validation split for every epoch. The output vector $\tilde{f}_m \in \mathbb{R}^{32}$ is computed according to

$$\tilde{f}_m = h(f_{m-1}, f_{m-2}), \tag{4.8}$$

where $h : \mathbb{R}^{\mathbf{32 \cdot 2}} \rightarrow \mathbb{R}^{\mathbf{32}}$ is the function computed by the autoencoder. Outlier detection is carried out based on the reconstruction error between a given frame and the output generated by the autoencoder. Figs. 4.29 and 4.30 illustrate the reconstruction errors for several frames of two illustrative sequences.

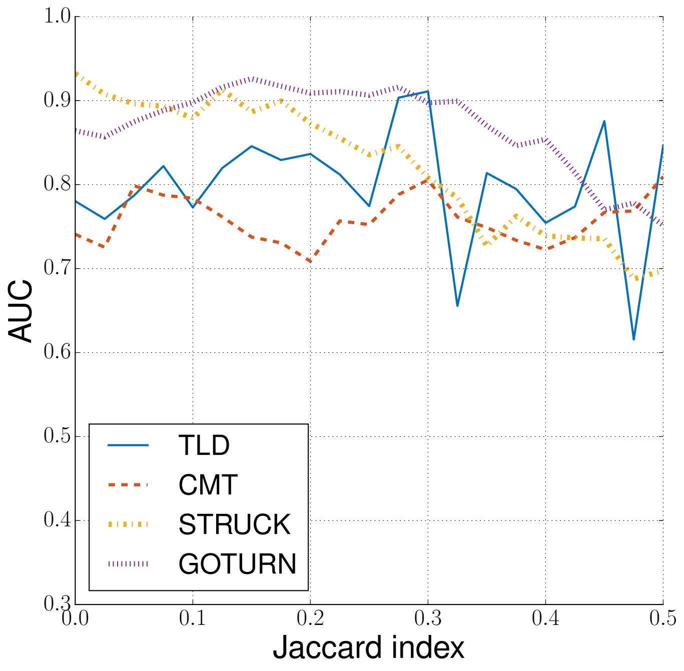


Figure 4.27: Area under the curve for the trackers for various values of τ using the method described in Section 4.3.3 and shown in Fig. 4.24. We note the Autoencoder is capable of more robustly detecting anomalies for higher τ.

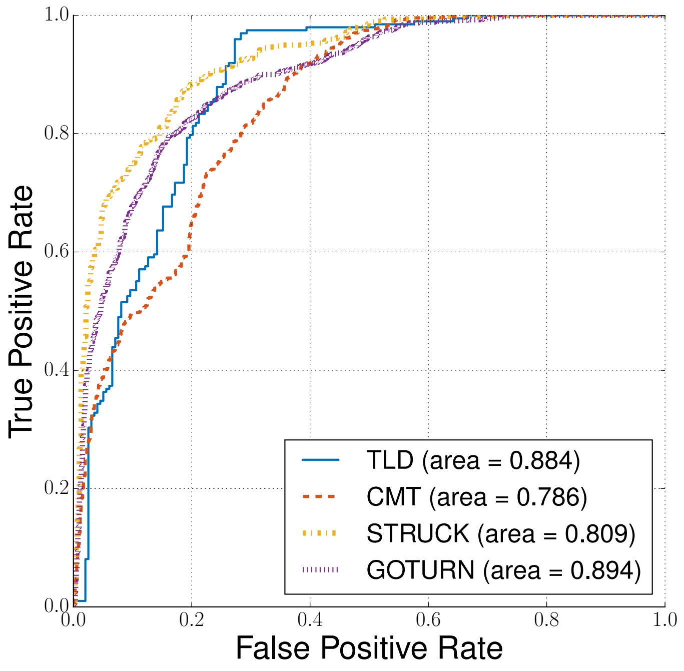


Figure 4.28: ROC curves for the Autoencoder method at $\tau = .3$ on a test set. We observe improvement in three of the trackers. The exception is GOTURN, for which the baseline approach in Figure.4.13 performs better.

Training the autoencoder is done by using stochastic gradient descent to find the parameters that minimize the mean squared error between the input and the output. With an input set of examples Ξ, our loss function is defined as

$$\Phi(\theta) = \sum_{f \in \Xi} \left\| f - \tilde{f} \right\|^2, \tag{4.9}$$

To assist our network in learning the temporal dependencies we used 1D convolution layers in a manner similar to Wavenet [80]. This changes our neurons to become the convolution of the input and weight matrix from the previous layer. As expressed in [38], we define our network activations for layer l for all v neurons as

$$\alpha(f^l) = \sum_{i=l}^{N_{l-1}} \sigma(W_{iv}^{l-1} * f_i^{l-1}) + \iota_v^l, \tag{4.10}$$

where $*$ is the 1-dimensional convolution operator and $\sigma(\cdot)$ is the non-linear hyperbolic tangent activation function. This is the same as Eq. 3.19, which is repeated here for convenience. An important consequence is that because of the convolution operator our hidden layers are 2D. The last flatten layer brings the shape back into 1D at the end of the architecture.

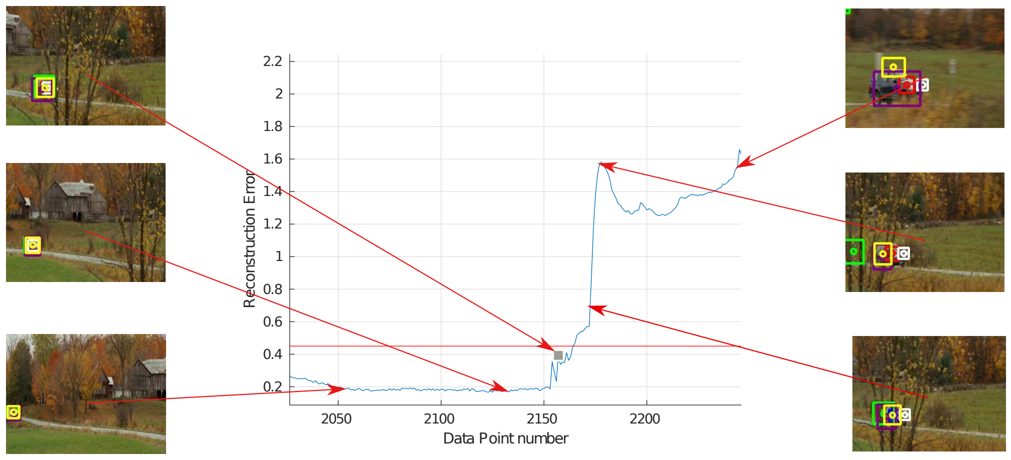


Figure 4.29: Illustration of the results generated using the proposed approach based on autoencoders for the sequence *Car*. See the caption of Fig. 4.30 for a description of the elements of the figure. We note that the reconstruction error from the autoencoder scales consistently with the expected confidence of the trackers.

We evaluate the ability of the method based on Mahalanobis distances as well as our proposed approach to detect outliers by computing the area under the receiver operating characteristic (ROC) curve (AUC) for values of the Jaccard index threshold τ between 0.0 and 0.50. As Fig. 4.12, which is shown for our baseline approach, and our proposed approach in Fig. 4.27 indicate, our proposed approach substantially outperforms the baseline method, particularly for TLD and STRUCK. Fig. 4.28 shows the ROC curves for each individual tracker for a fixed Jaccard index threshold of $\tau = 0.30$. That is, the area under the curve in these figures correspond to the point with Jaccard index $\tau = 0.30$ in Fig. 4.27. As the figures demonstrate, our method outperfoms the baseline approach in most

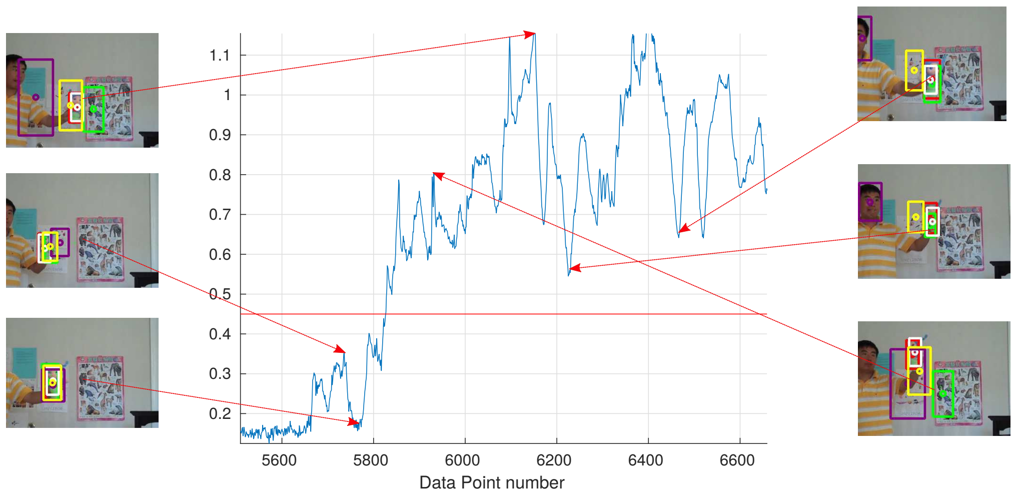


Figure 4.30: Illustration of the results generated using using the proposed approach based on autoencoders. The images show snapshots of several frames in the sequence *Doll*. The red, green, white and purple boxes correspond to the outputs of TLD, CMT, STRUCK and GOTURN respectively. The yellow box is the output of the fused approach. The graph in the center shows the values of the reconstruction errors for the corresponding frames. We can see that higher reconstruction errors are associated with higher levels of anomaly. We also note that the reconstruction error decreases when the trackers get closer to object of interest.

cases, with a gain of approximately 70% for STRUCK. As shown in Figs. 4.14 and 4.15, the supervised approaches described in Section 4.2.5 fail to generalize and do not provide satisfactory results for most trackers under the same conditions. Qualitative results are shown in Figs. 4.29 and 4.30 for our method and in Fig. 4.11 for the baseline approach based on Mahalanobis distances. As evident by our results, we have reason to believe our network to differentiate anomalies in a manner that is comparable or better to the baseline approach.

4.3.5 Chapter Summary

Through our experiments, we determined that the autoencoder and particularly the convolutional autoencoder was the most capable at detecting anomalies in tracking data. This was done using a uniform test that we defined as our problem statement. Given a set of data from our tracker ensemble, we split this data into normal and faulty data. Next, we used ROC curves to compare various anomaly detection techniques. Our results showed that autoencoders performed favorably relative to our benchmark. This opens up the possibility of using the autoencoder as a replacement for the Mahalanobis Distance metric. By acquiring this network that can detect anomalies we plan to apply a variation of this method unto our tracker ensemble and use it as a scoring mechanism in the next chapter.

CHAPTER 5

BAYESIAN AUTOENCODER MAXIMUM LIKELIHOOD DATA FUSION

In the previous chapter, we proved that it was possible to detect anomalies in the easy scenarios where either all the trackers were keeping track of the target or they were all lost. While this is useful in some scenarios, we found that was not ideal for our application as the majority of our frames do not fall into this category. Mathematically speaking, the formulation in Eq. 4.2.2 meant that only approximately 30% of the data would be acceptable and 70% of our data was anomalous. However, we still need a mechanism to determine the confidence of the trackers for these 70% of cases.

Thus, it became necessary to reformulate our problem. To do this we propose changing our approach to handle the outliers of individual trackers so it could provide a better weighing than the Kalman filter. Rather than looking at our network as a mechanism that can detect failures from any of the trackers, we propose to develop multiple networks that can learn to differentiate the failures from the individual trackers. That is, we propose a method that creates an anomaly detection mechanism for every single tracker.

5.1 Proposed Network Architecture

We selected the two-dimensional convolutional autoencoder as our network for anomaly detection scoring. However, because of the change to our problem statement, this required modifying our outlier and inlier datasets as well as our input vector in Eq. 4.1. Due to the benefit of using multiple frames, we employed the paradigm introduced in Eq. 4.8.

5.1.1 Network Description

To train these new networks, we find the frames in which that tracker is non-anomalous. This is inherently a significantly more challenging problem for the networks, since there is less information about the performance of the other trackers and the network cannot use those as references. Since this makes our data inherently more chaotic, this becomes a tougher problem.

The first network we use is based on the network in Section 4.3.3, where we found that convolutional autoencoders generate the best results. However, rather than using 1D convolutions, we modified our input vectors to be a 2D input. This allows the convolutional filters to better learn interrelationships between the various trackers by performing the convolution operation on them together.

Through experimentation, we found a new method was superior to our original proposal. Rather than creating a $1 \times Nm$-dimensional vector for every frame, we generated a $N \times m$ matrix using the output from the trackers according to

$$f_m = \begin{bmatrix} x_m^{(1)} \\ x_m^{(2)} \\ , \ldots , \\ x_m^{(N)} \end{bmatrix} . \tag{5.1}$$

Similarly to our previous network, we use two previous frames to predict the current frame. For our four trackers the output $\tilde{f}_m \in \mathbb{R}^{32}$ is computed according to

$$\tilde{f}_m = h(f_{m-1}, f_{m-2}), \tag{5.2}$$

where $h : \mathbb{R}^{\mathbf{32 \cdot 2}} \rightarrow \mathbb{R}^{\mathbf{32}}$. Due to the benefit of using multiple frames, we employed the paradigm introduced in Eq. 4.8 and repeated in Eq. 5.2 for convenience. Fig. 5.1 illustrates our final network topology.

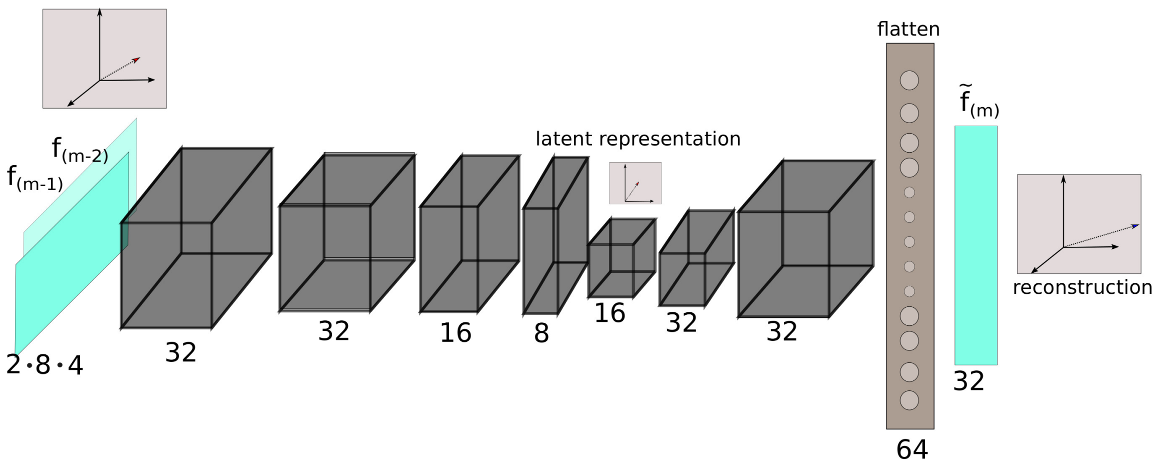


Figure 5.1: Final autoencoder model.

5.2 Data Partitioning and Network Training

To train our network, we partition the data similarly to the previous section.

1. When one tracker at a frame has a Jaccard index less than τ we consider this "anomalous" data for that specific tracker. This also creates N different datasets (one for each tracker),

$$\mathcal{F}_{e^{(k)}} = \left\{ f_m \in \mathcal{F} | J(b_m^{(n)}) < \tau \right\}, \tag{5.3}$$

for $n = 1, ..., N$ and for $m = 1, ..., M$.

2. When one tracker at a frame has a Jaccard index greater than τ we consider this "normal" data for that specific tracker. This also creates N different datasets (one for each tracker),

$$\mathcal{F}_{l^{(k)}} = \left\{ f_m \in \mathcal{F} | J(b_m^{(n)}) > \tau \right\}, \tag{5.4}$$

for $n = 1, ..., N$ and for $m = 1, ..., M$. This is also equivalent to the compliment of $\mathcal{F}_{e^{(n)}}$.

To act as a better weight for trackers with a success rate that shows that trackers only have keep track of the target than 50% of the time [84]. This would imply that in most videos there are very few scenarios where all trackers are operating according to Eq. 4.3. This implies that we need autoencoders that are more general for our tracker weighing application and more suitable for the much more likely situation that at least one tracker is lost. In Eq. 5.3, we demonstrate how we acquire training data for the problem of creating weights for the tracker. As before, we partition our data into training and test sets, $\mathcal{F}^1$ and $\mathcal{F}^2$ respectively. This partition is done for every tracker, so that every tracker has a local classifier.

We trained the network according to topology described above. We first found the frames for which tracker n $J > \tau$. This was done for datasets $\mathcal{F}^1$ and $\mathcal{F}^{aux}$. With $\mathcal{F}_l^{(n)}$ from the training set and $\mathcal{F}_l^{(n)}$ from an additional training set for all N, the training data was ready. Each frame was prescaled according to Eq. 4.6. Next for each of the N trackers a network was generated according to the model in Fig. 5.1. Each of the k networks was trained with the Adam optimizer for 15 epochs where k is the network corresponding to tracker n, and with uniform Gaussian noise applied to the input. Next, each network was deployed on the "anomalous" and "normal" data (as define above), $F^1_{l^{(n)}}$ and $F^1_{e^{(n)}}$. The reconstruction error as defined in Eq. 4.9 was computed for each input frame pair f_m and prediction $\tilde{f}_m$. In Fig. 5.2, we see that for all K networks for all N different trackers, outliers have a higher mean reconstruction error value.

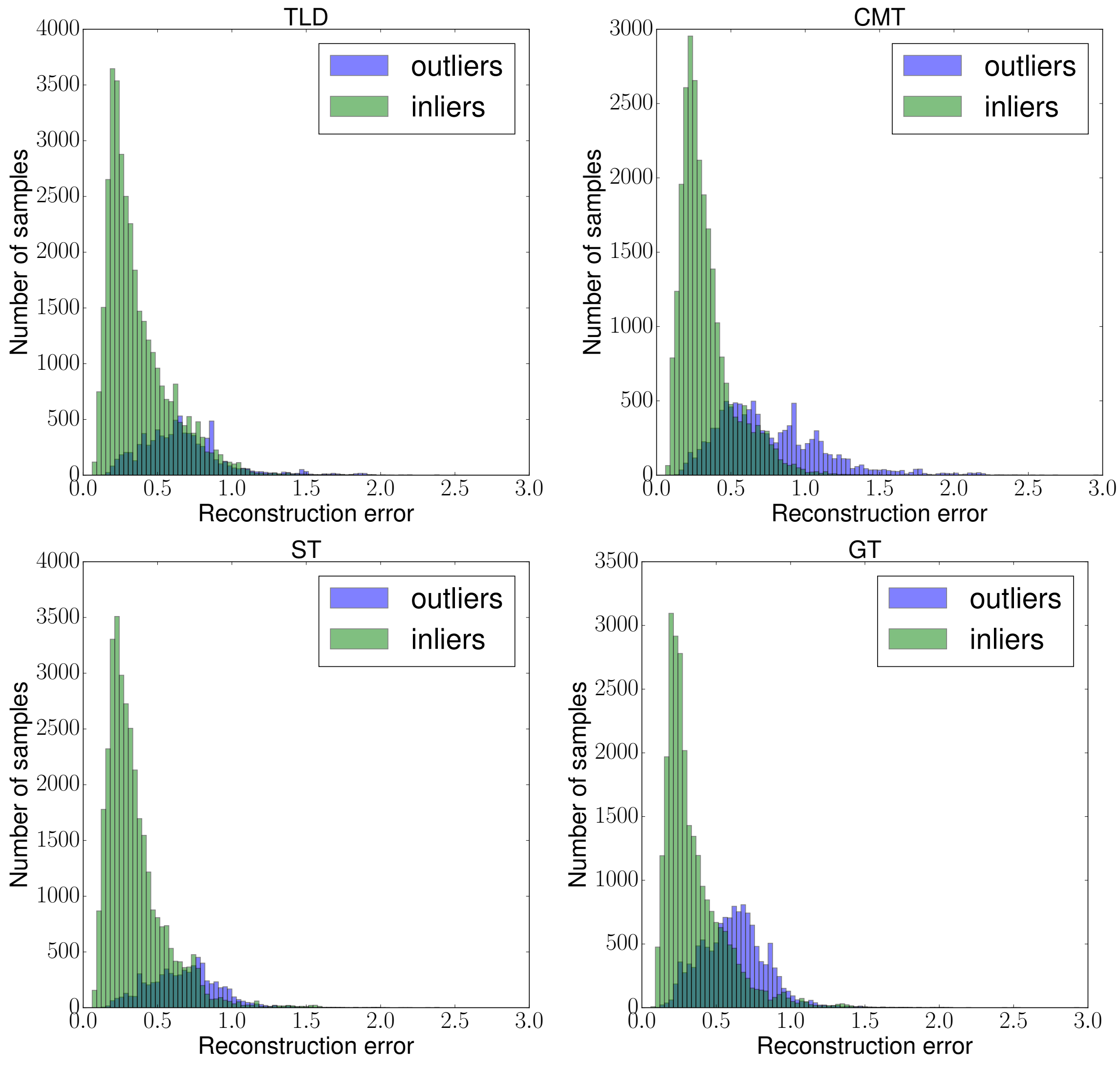


Figure 5.2: Histograms demonstrating reconstruction errors for all k trackers on the training set on $\tau = .3$. The outliers $F_e^{(k)}$ are blue and inliers $F_l^{(k)}$ are in green. Best viewed in color.

5.2.1 Network Results

We present the anomaly detection results of each of the four trackers for both of our networks. To confirm that our network was not overfitting on the training set $\mathcal{F}^1$, we applied our model on a different set, but with the same partition rules as in Eqs. 5.3 and 5.4. After training each network, we observed the success with which that method was capable of extrapolating its performance on $\mathcal{F}^2$. In Fig. 5.3, we see that not only are our models successful at differentiating between outliers $\mathcal{F}_{e(n)}$ and inliers $\mathcal{F}_{l(n)}$, but they are able to do this without any prior knowledge of $\mathcal{F}^2$. Another interesting note is that the means for $\mathcal{F}_{l(n)}$ for both datasets are similar. This implies that the networks have learned an effective model for the inlier data that is consistent between different datasets.

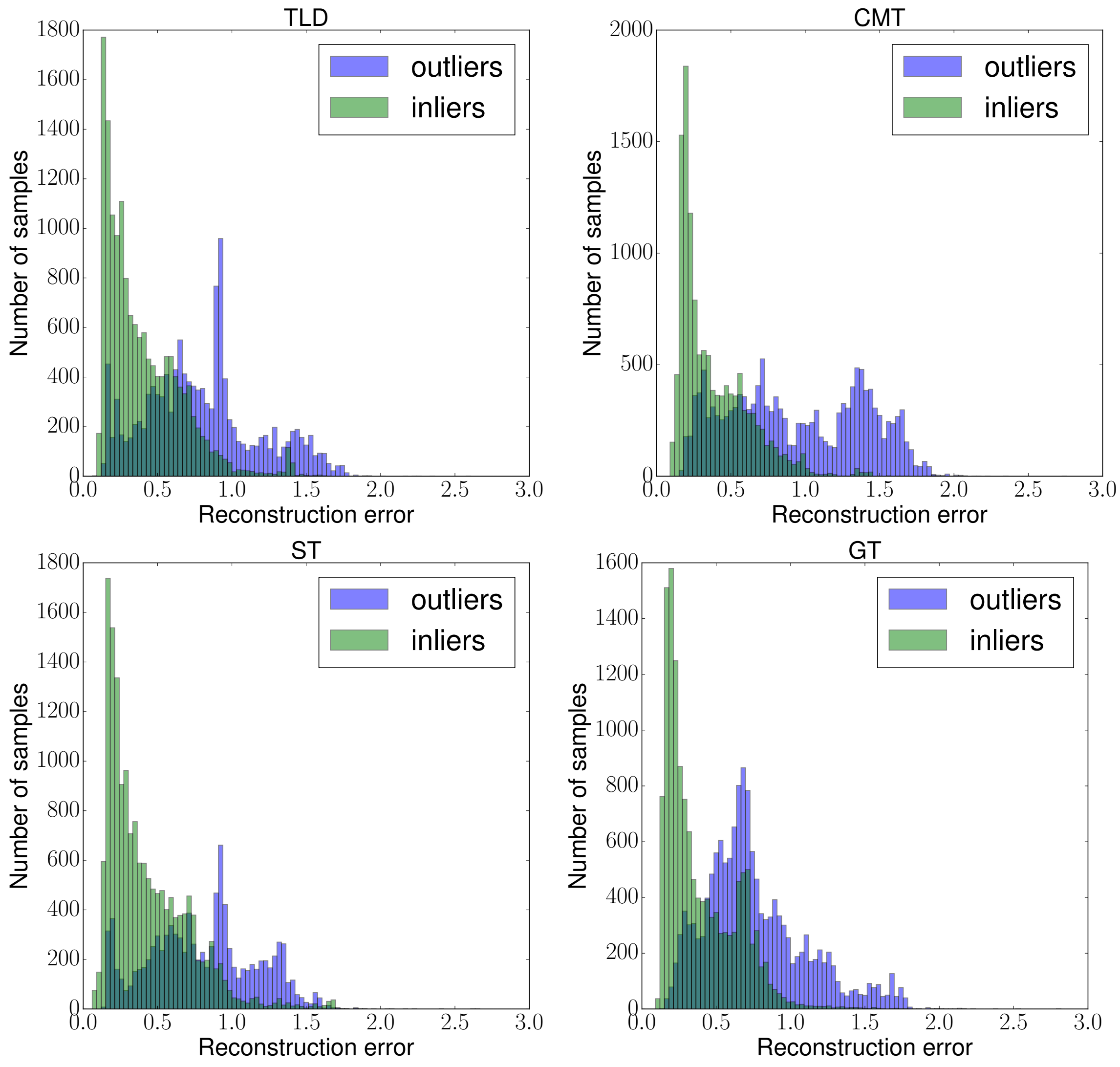


Figure 5.3: Histograms demonstrating reconstruction errors for all k trackers on the test set on $\tau = .3$. The outliers $\mathcal{F}^2_{e(n)}$ are blue and inliers $\mathcal{F}^2_{l(n)}$ are in green. Best viewed in color.

We notice that while the mean reconstruction error and standard deviation are higher on the test set, results are still comparable to the training set. More importantly, these results demonstrate that in both cases the network is able to learn a statistical representation of what constitutes "positive" and "anomalous" data as shown in Fig. 5.2 and Fig. 5.3. By acquiring data from normal operation, we see that our network creates a statistical approximation for what constitutes that in a data independent sense. With a larger data source it is likely that performance will become even more generalized.

To quantify the performance of each of our networks we compare the ROC curves for all K networks. We notice in Fig. 5.4 that the performance of our trackers is relative consistent with the work in Chapter 4, albeit with results that are slightly worse due to the more challenging nature of this updated problem.

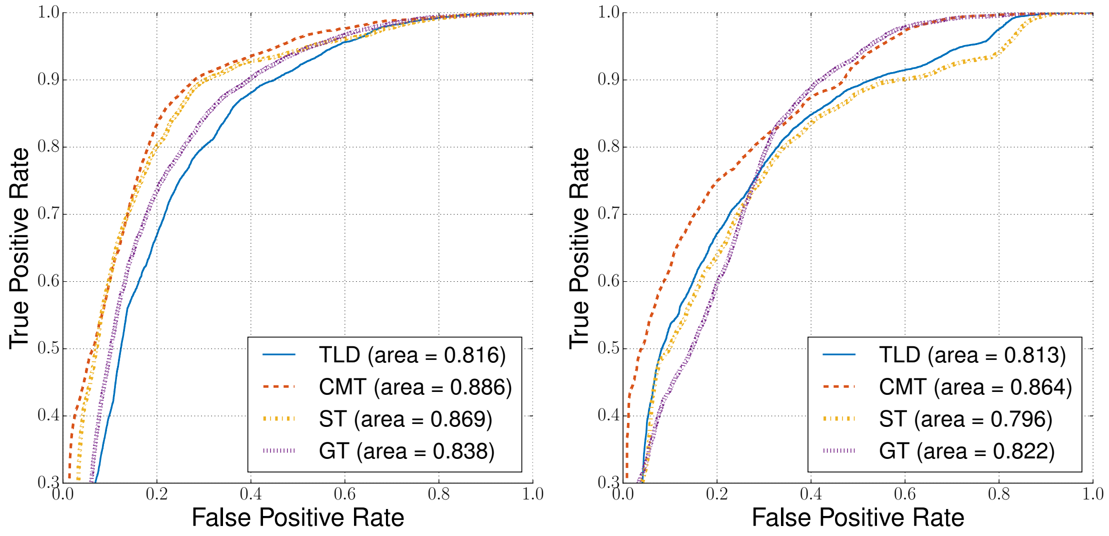


Figure 5.4: ROC curves comparing the performance of the k trackers with $\tau = .3$. The left graph demonstrates performance on the training set $\mathcal{F}^1$ and graph on the right corresponds to the test set $\mathcal{F}^2$. This ROC is different in that each tracker has a dedicated network

Additionally, further research suggested to examine the precision recall curves to get an accurate representation on performance with unbalanced classes [14]. A precision recall curve was generated naturally from the data in the histograms. We see in Fig. 5.5 that our method is capable of detecting outliers based on this other metric as well. Additionally, we note that our method performs better on the test set than on the training set based on this metric according to Fig 5.5. This is perhaps due to some issues with false positives that the training set faces,

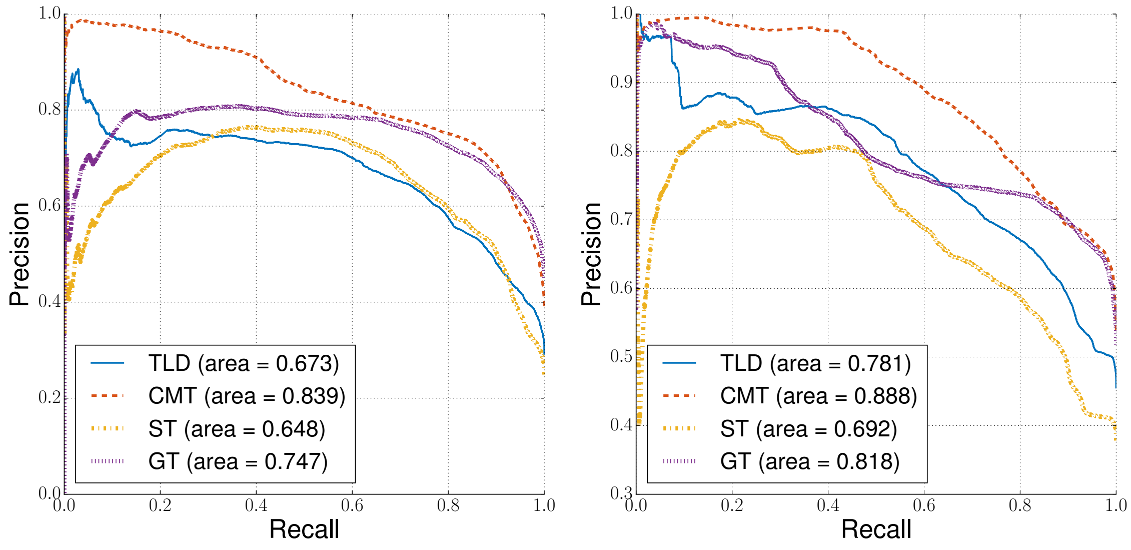


Figure 5.5: Precision recall curves comparing the performance of the n trackers on $\tau = .3$. The left graph demonstrates performance on the training set $\mathcal{F}^1$ and graph on the right corresponds to the test set $\mathcal{F}^2$.

5.2.2 Comparison to Baseline

Here we explore the precision recall curves for the baseline Mahalanobis distance method. We acquire the values according to Eq. 4.5 for each local tracker and generate precision recall curves in using the partition described in Section 5.2.

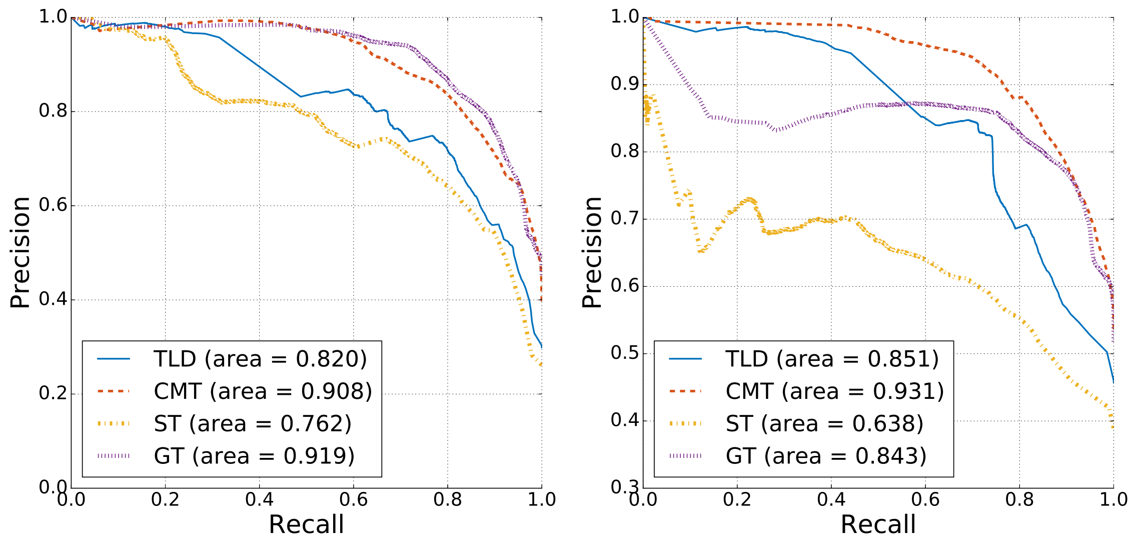


Figure 5.6: Precision recall curves comparing the performance of the n trackers on $\tau = .3$ using the Mahalanobis distance metric. The left graph demonstrates performance on the training set $\mathcal{F}^1$ and graph on the right corresponds to the test set $\mathcal{F}^2$.

We see in Fig. 5.5 that the precision recall scores are better in our method for STRUCK, but the Mahalanobis distance still outperforms our method for the other trackers. This can be seen by comparing the results in Fig. 5.6 to Fig. 5.5.

To improve upon the performance of our method, we propose weighing based on a log maximum a posteriori score. Additionally, we propose an Offset that reduces the bias associated with the $0th$ degree moment of our distribution. More concretely, we introduce an adaptive offset that is based on a sliding window similarly to the work done in [33].

5.3 Maximum A Posteriori Score

To get better results for our anomaly detection mechanism we propose a maximum a posteriori (MAP). The purpose of this score is to bring our scoring into a statistical framework. Therefore our reconstruction error will be used in a manner that is "Bayesian", which keeps with the original goal of this contribution.

To do this, we first formulate the probability of our score being an outlier based on the probability $P(O_n|\varrho_k)$, which is a representation of the probability that tracker n is currently in an anomalous state for a given reconstruction error. Using Bayes rule, we can rewrite the function as

$$P(O_n|\varrho_k) = \frac{P(\varrho_k|O_n)P(O_n)}{P(\varrho_k)}, \tag{5.5}$$

In our approach, we define $P(\varrho_k|O_n)$ as the probability that the n-th tracker is in an anomalous state given a certain reconstruction error. To calculate this, we use the histogram shown in Fig. 5.2 on the training set $\mathcal{F}^{(1)}$. To classify a point, we must first generate a distribution summarizing the likelihood of being an outlier given a certain reconstruction error [12]. By binarizing the results into 30 distinct bins, we generate a table of outlier probabilities for reconstruction errors between 0 and 3 with a step size of .1.

$P(O_n)$ is a static probability based on the tracker being accurate. Since this is largely impossible to know entirely correctly, two approximations are possible here. An assumption of .50 is possible, which would imply the tracker has an equal chance of being correct or incorrect, this would be beneficial where we have no prior information on how well that tracker will perform in a particular scenario. In situations where we know how well the tracker performs, such as the OTB benchmark, a probability based on the performance in that scenario exists. We chose the second assumption and based $P(O_n)$ on the precision score of the tracker.

Similarly to the formulation for $P(O_n|\varrho_k)$, it is also possible to approximate the probability that your current tracker is operating normally.

$$P(I_n|\varrho_k) = \frac{P(\varrho_k|I_n)P(I_n)}{P(\varrho_k)}, \tag{5.6}$$

We define $P(\varrho_k|I_n)$ as the probability that the n-th tracker is in an non-anomalous state given a certain reconstruction error where $P(I_n)$ is the complement of $P(O_n)$. Similarly to the approach above, we use the bins for the “normal” class in the training set to generate a probability distribution for this set. We demonstrate an example of what the inliner and outlier probabilities would look like in Fig. 5.7.

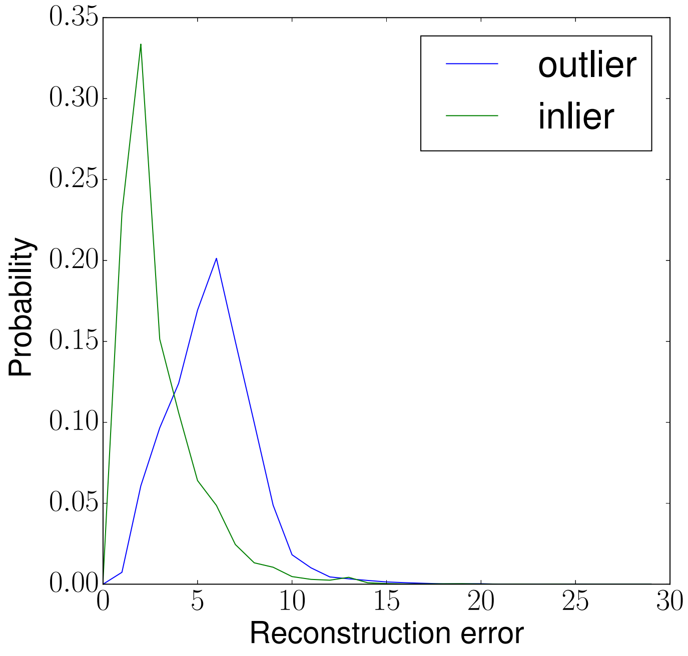


Figure 5.7: Example of the probability distribution for the positive and negative classes generated where green corresponds the positive class and negative class is represented as blue. Best viewed in color.

Now, we have two probabilities of whether our sample is an inlier or an outlier for a specific reconstruction error. To determine in which of the two states the tracker is, we use a log-maximum a posteriori (MAP) similar to the work in [74] to calculate the log posterior distribution

$$\mathcal{P}_n = \ln \frac{P(O_n|\varrho_k)}{P(I_n|\varrho_k)}, \tag{5.7}$$

which becomes

$$\mathcal{P}_n = \ln \frac{P(\varrho_k|O_n)P(O_n)}{P(\varrho_k|I_n)P(I_n)}, \tag{5.8}$$

Normally, in a similar log likelihood ratio test, if the value from the test is above a certain positive threshold, it would be classified as an outlier. However, in our case the fact that this log posterior distribution ratio generates a score is further used by our approach. We use the score from this maximum a posteriori for a specific sample in a manner similar to how the Mahalanobis distance is used in Eq. 3.4. Like the MD distance, we must then transform this score into a quantity that the covariance matrix can use, which we explain in the next subsection.

5.3.1 Reconstruction Error as Source of Information

By looking at the log posterior distribution scores, we can use this information is in a manner similar to the Mahalanobis distance metric. We can generate weights for our trackers in the ensemble based on a probabilistic representation of how close the sample is to our "normal" sample distribution.

We propose a smoothing function similar to Eq. 3.8 for each of the maximum a posterior log likelihood (MAP) scores $\mathcal{P}_n$ for each of the N trackers in our ensemble,

$$w_a = \rho_0 + \rho * \tanh(\kappa * (\mathcal{P}_n - \psi)). \tag{5.9}$$

Here ρ_0 represents the vertical displacement and ρ corresponds to the amplitude of the hyperbolic tangent function. This places the bounds of our output between $\rho - \rho_0 \leq w_a < \rho + \rho_0$. κ is the slope with which penalization takes place, a higher κ would mean the function transitions much more abruptly between the extremes, whereas a smaller κ means that the transition is smoother. These three variables act as parameters which we estimate heuristically based on the performance of the trackers on the visual tracking benchmark. Finally ψ

represents a moving horizontal offset that attempts to compensate for the zeroth moment drift. We present this in Algorithm 2. We use a moving average that updates every λ frames, for our results we chose 2 as our λ value.

Algorithm 2 Calculate moving offset.

```
Input: Set of $n$ trackers $s_j \in \mathbb{S}$, initial bounding box $x_0$, set $V$ of images, stack $\varepsilon_k$
    for every tracker
Output: Horizontal offset $\psi$
 1: Initialize $\psi$ as $\varrho_{f_0}$.
 2: while $V$ has new images do
 3:     Load new image
 4:     for Each tracker $s_j \in S$ do
 5:         Generate bounding box $x_j$ for each tracker $s_j$
 6:         Generate reconstruction error for each tracker $\varrho_k$
 7:         push to stack $\varepsilon_k$
 8:     end for
 9:     Wait for all trackers $s_j$
10:     if $mod(V, \lambda) == 0$ then
11:         Generate temporary variables $\vartheta_k$ for all $k$
12:         for Each tracker $s_j \in S$ do
13:             Sum up 2 most recent values and pop from $\varepsilon_j$
14:             Divide by 2
15:             Set that value to $\vartheta_k$
16:         end for
17:     end if
18:     Set $\psi$ as the maximum of $\vartheta_k$ divided by 2
19: end while
```

We summarize our proposal for using the reconstruction ϱ to generate a score in Fig. 5.8.

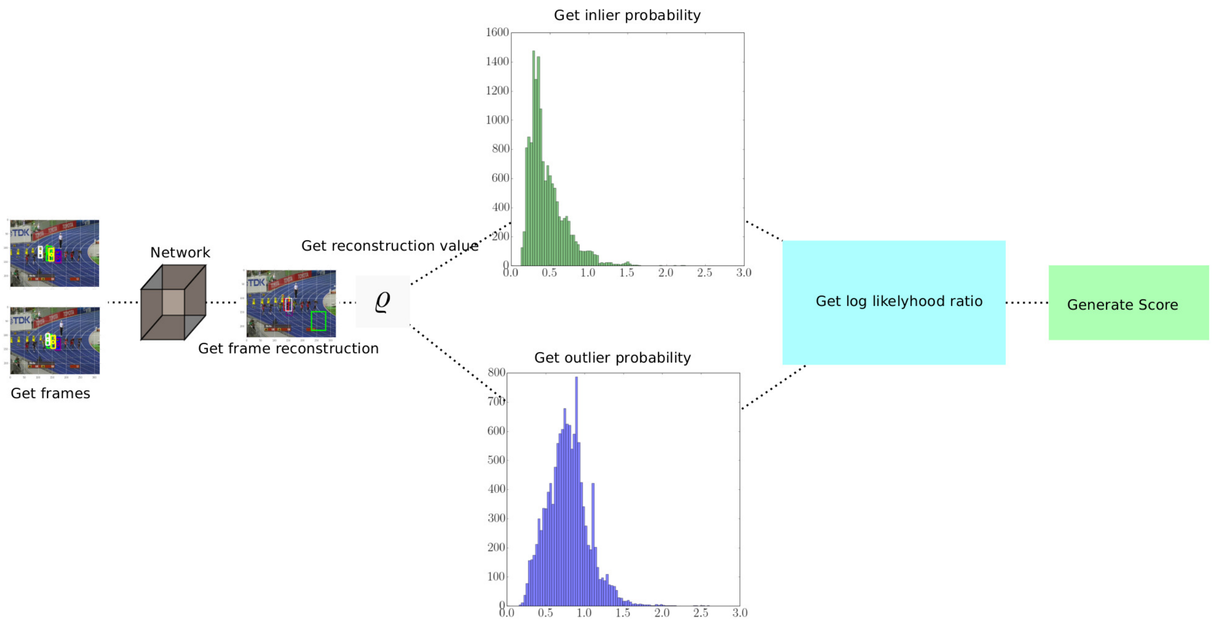


Figure 5.8: Summary of our reconstruction error based scoring approach. Best viewed in color.

Now that we have a model for determining whether a sequence is an outlier or an inlier, we perform different simulations to determine that our method is valid. We propose three experiments that examine what our method does on the $\mathcal{F}^{(1)}$ dataset. In the next section we set up a number of various simulation exercises to check our method for logical consistency. In particular, we wanted to make sure that this method performed in a reasonable manner offline before applying these networks online as part of the tracker ensemble.

5.3.2 Offline Results

We first examine the performance of our method offline on the *bolt* image sequence. We choose this sequence because in the previous work with Hierarchical Bayesian Data Fusion, this method was particularly challenging. Our goal is to affirm that our method can tackle this tough sequence. We present select frames from this example in Fig. 5.9.

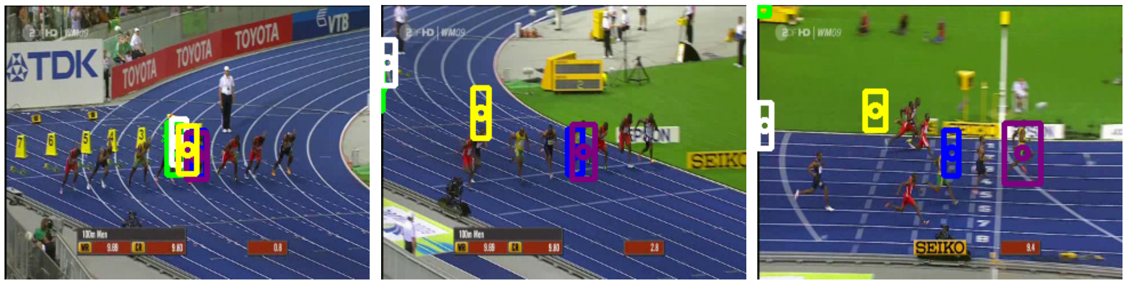


Figure 5.9: Illustration of the performance of the $n = 4$ trackers and the previous global estimate on the *bolt* sequence. Refer to Fig. 4.3 for a description of the various bounding box colors. Best viewed in color.

In *bolt*, we see that at the beginning all the trackers are capable of following the target. Eventually, CMT and STRUCK both become lost. After which eventually, TLD and GOTURN became lost as well. In the middle of the sequence, TLD was technically on the target by staying in place but the tracker was lost. Throughout most of the sequence, the correct course of action would be to trust GOTURN, however this does not happen. Due to these reasons, this sequence was a difficult example for our approach to tackle.

First, we deploy our models and examine the reconstruction error ϱ for our various trackers on the *bolt* sequence. In Fig. 5.10, we see that GOTURN has the lowest reconstruction error and the smallest drift, which implies that its network displays the most confident in this tracker.

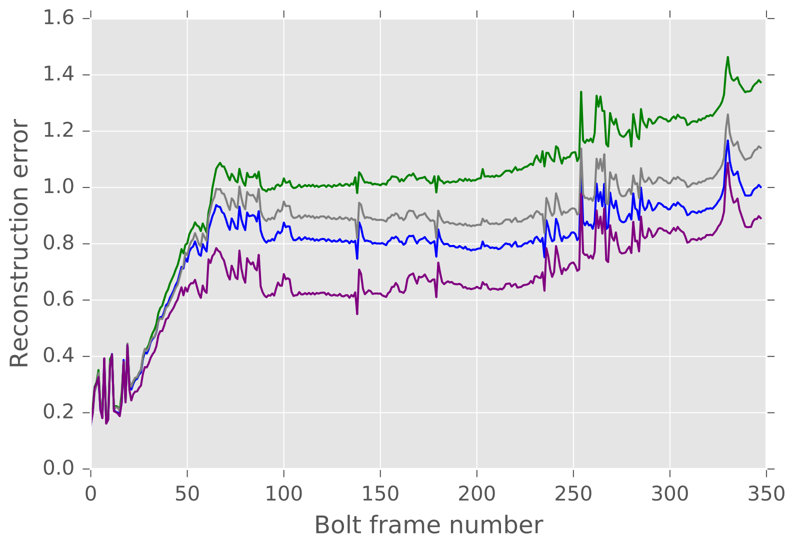


Figure 5.10: Reconstruction error ϱ. Blue, green, grey and purple correspond to TLD, CMT, STRUCK and GOTURN, respectively. Best viewed in color.

Next, we apply the transformation in Eqs. 5.5-5.7 to generate a score metric. We plot this in Fig. 5.11, and the results remain consistent but now there is information to imply that the network becomes more confident in STRUCK at the end, which is an error. Additionally, our score values in Fig. 5.11 properly classifies CMT as lost. In this sequence, this is the proper course of action because CMT completes fails to follow the target and the score reflects that.

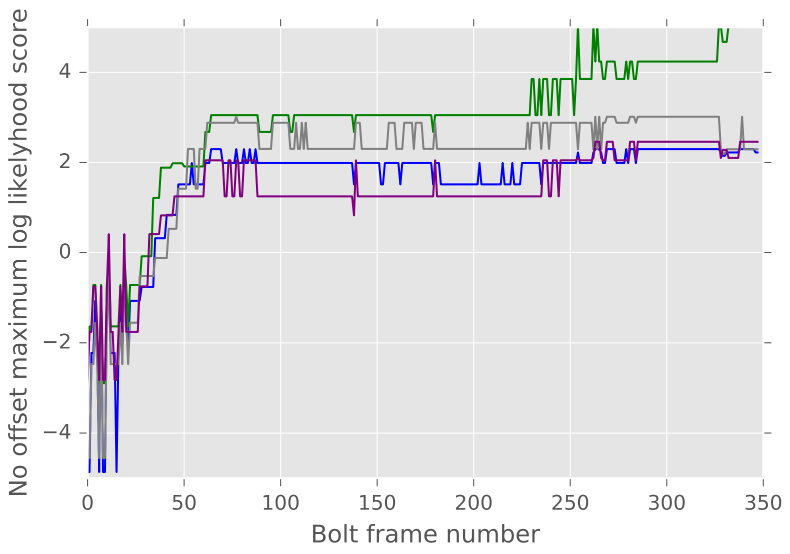


Figure 5.11: $\mathcal{P}_k$ for all k trackers TLD, CMT, STRUCK and GOTURN in blue, green, grey and purple respectively. We can see that GOTURN has the highest amount of trust, but we also observe a drift up in value. This motivates our inclusion of an offset. Best viewed in color.

We show the change made by applying our offset function to $\mathcal{P}_k$ we are able to get a much better input into our function that will predict a lower score for GOTURN. In Fig. 5.12, we see this result and see that the network generally trusts GOTURN the most, followed by TLD. The network completely disregards CMT here, which is the proper course of action. These results demonstrate a sensibility in the result and acts as sanity check to show that this method can be ported to real-time application as a part of the tracker ensemble in the OTB framework. The exception to this sensible operation is in frames $230 - 310$, but this corresponds to when GOTURN also begins to lose track.

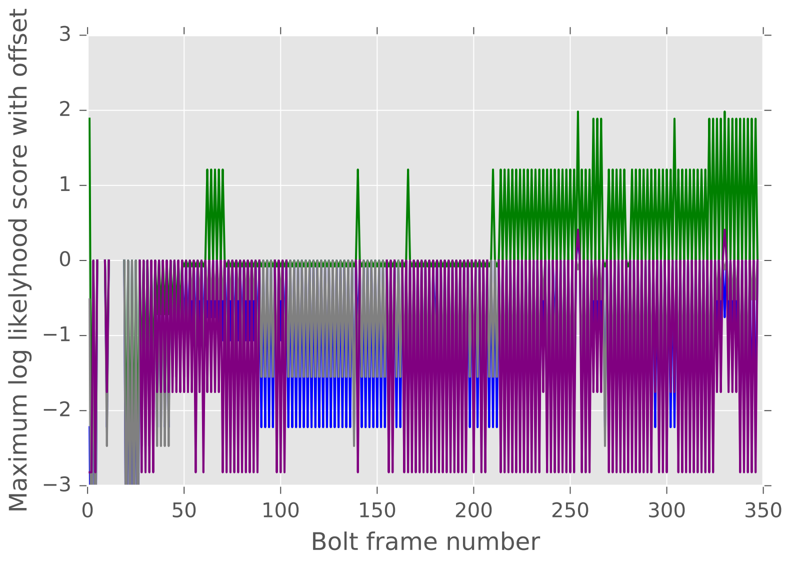


Figure 5.12: $\mathcal{P}_k$ with the offset for all k trackers TLD, CMT, STRUCK and GOTURN in blue, green, grey and purple respectively. We see the benefit provided by the offset by shifting GOTURN towards -1 which would imply that our algorithm completely trusts the tracker for those frames. Best viewed in color.

Lastly, to further test that the performance is indeed reasonable, we wanted to quantitatively evaluate a larger dataset. To do this we evaluated our MLL score on the whole training set. We found the average $\mathcal{P}_k$ with the offset for all k trackers for the whole $\mathcal{F}^{(1)}$ dataset. Our interest was to see what the box plot values would be for various Jaccard values τ. Ideally, our method would score all low τ values as 1, and score all high τ values as -1. This would imply a perfect classification. However, our box plot in Fig. 5.13, proves that while it is not perfect, our network tends to classify results with a Jaccard value $< .3$ properly as outliers and $> .3$ properly as inliers.

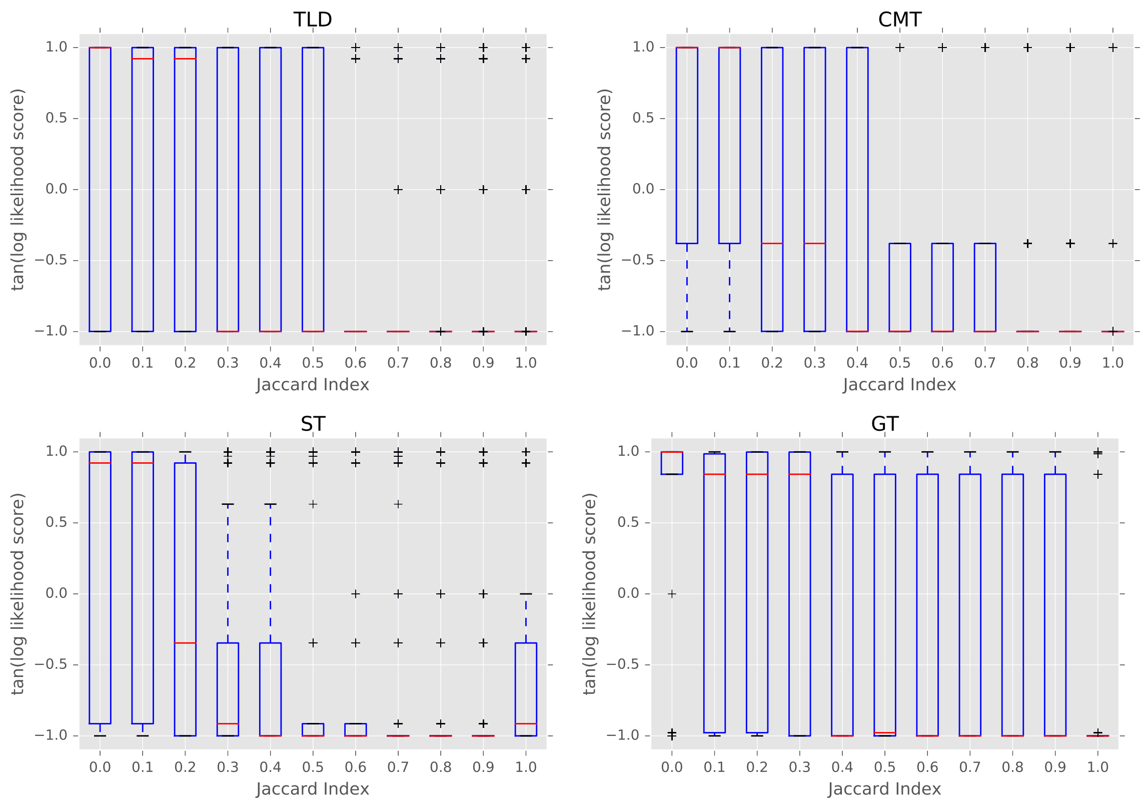


Figure 5.13: Box and whisker plot showing the distribution of the tangent of the likelihood $\tanh(\kappa * (\mathcal{P}_k - \psi))$ for various τ of the ground truth for all k trackers; TLD (top left), CMT (top right), Struck (bottom left) and GOTURN (bottom right). The red line represents the median for that specific τ value, the stars represent outliers in the data. The boxes correspond to the 1st and 4th quartile, the whiskers correspond to 1.5 multiplied by the interquartile range. In general, for smaller τ, our method predicts the outlier and for higher τ it correctly tends to predict that the object is an inlier.

From our results, we see that our method is able to capture the difference between outliers and inliers for the different sequences. In Fig. 5.13, we see that for $\tau > 0.30$, our method on average is equal to -1 as shown on the red lines on the box plots. These results are consistent with what we expect our method to accomplish. We also see that for STRUCK our method has the weakest performance. This simulation validates our method as sensible to apply to the real-time tracker application. This allowed us to proceed and integrate the networks into the original HABDF framework.

5.4 Application to Tracking using Hierarchical Bayesian Data Fusion

We propose Bayesian Autoencoder Maximum A Posteriori Data Fusion (BAMAPDF), an approach that uses our autoencoder-based maximum a posteriori score to weigh the trackers. We base this method on the HABDF, but modify it by substituting the MD weight with our autoencoder MAP based approach.

Our method uses the autoencoder maximum a posteriori proposed in the section above and specifically Eq. 5.9 to generate a weight for each tracker in our ensemble. We modify the original framework in Section 3.1 by modifying Eq. 3.9. Specifically we modify $R_{\sigma\sigma}$ to be a function of our weight from the autoencoder maximum a posteriori w_a

$$R_{\sigma\sigma}(w_d, w_a) = \Gamma w_d + \Delta w_a, \tag{5.10}$$

where Γ remains defined as before and we modify $\Delta = diag(\delta_1, \delta_2, \cdots, \delta_n)$ to represent the new diagonal matrix whose elements are the new function parameters. As before, γ_i and δ_i are set to 1 if there is no a priori knowledge of the system, we found that the values are closely linked to ones used in the previous iteration of this algorithm.

Otherwise, everything else is consistent and similar to the work in HABDF. We would also like to add that because the network is not very deep, the speed of each prediction of the autoencoder networks is less than 0.03 seconds in addition to the speed of the original algorithm, which can be considered real-time for most applications. Further computational performance improvements should be possible by optimizing our implementation with each prediction done in separate threading as well as other common techniques that take advantage of the Cuda library.

Similarly to the first version of this ensemble, the majority voting weight w_d and the autocoder maximum a posteriori (AMAP) weight w_a are used by the global tracker to update $R_{\sigma\sigma}$, which is then used in the global correction stage of the Kalman filter to generate the fused bounding box x_f.

We present our final approach in Fig. 5.14.

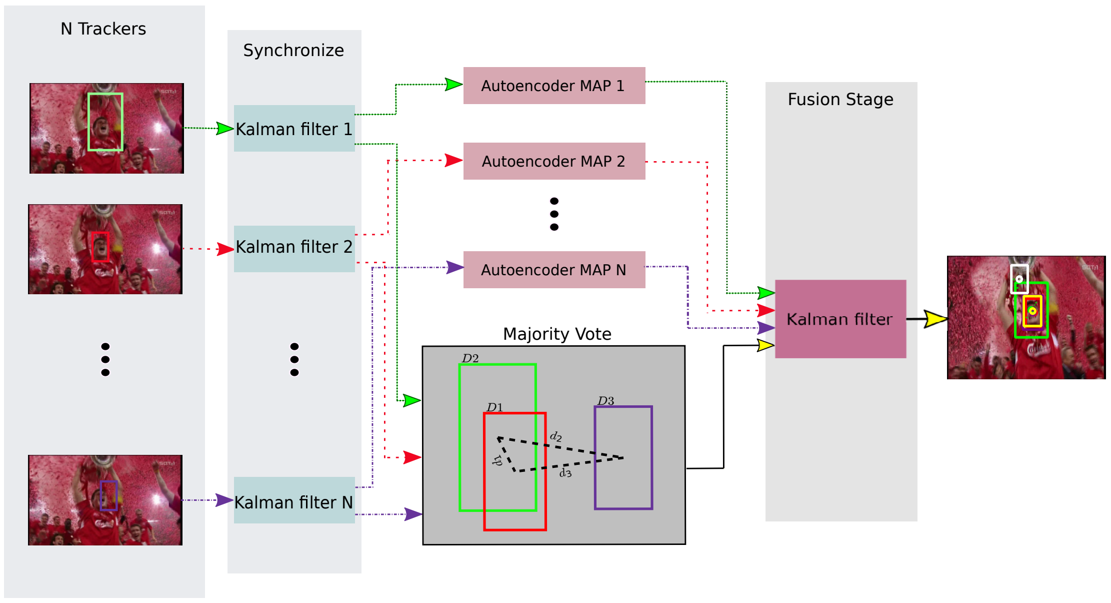


Figure 5.14: Schematic representation of the implementation of the proposed framework. Note how the Kalman filter blocks from Fig. 4.1 are replaced by our autoencoders. Best viewed in color.

5.4.1 Qualitative Results

Here we show qualitatively that our method offers certain benefits when compared to the prior approach. Initially, our motivation for using a machine learning approach was to tackle an intrinsic limitation of Kalman filter-based approaches when a tracker remains lost for a long time. Namely, after being lost for a long time, the tracker would stop noticing that it is lost, and begin to determine that it is correct due to the Markov chain assumption of the KF. In certain cases, we observe that our method can tackle this problem by learning a quasi-statistical representation of what likely constitutes a properly functioning tracker by learning a representation for an average of what a properly functioning looks like mathematically. We deploy the model and use the reconstruction error to see how far a frame is from this model.

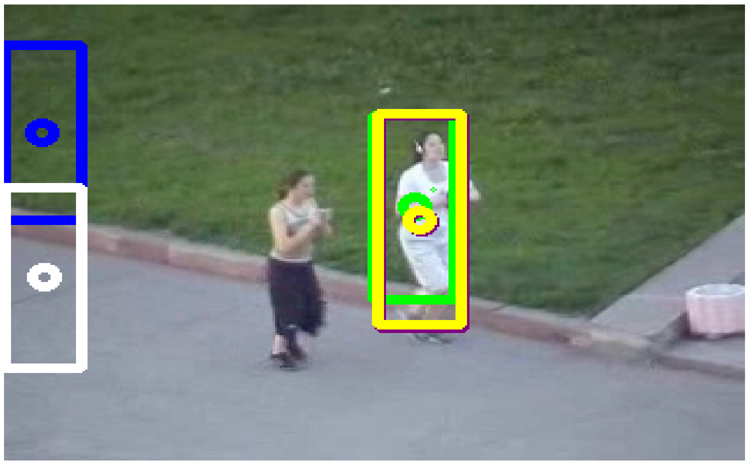


Figure 5.15: Scenario illustrating the case when only the two worst trackers in the ensembles are correct, whereas the generally superior TLD and STRUCK are completely failing. This figure shows the benefit and practical significance of our algorithm. A design choice based on the best all-purpose tracker might not have represented the most effective solution. See the caption of Fig. 5.16 for a description of the elements of the figure. Best viewed in color.

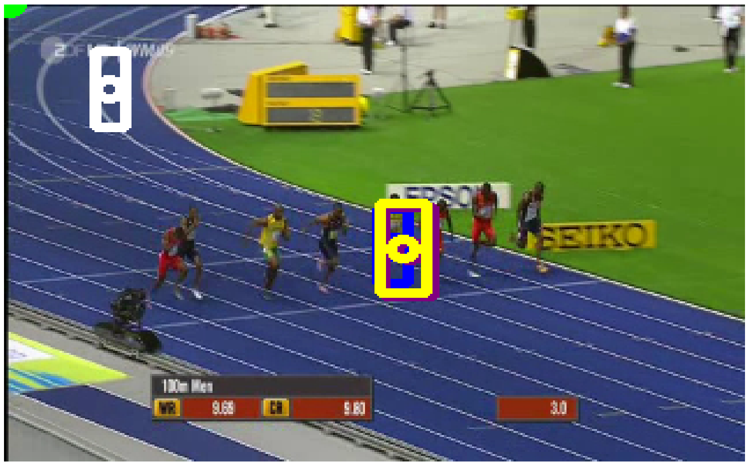


Figure 5.16: Success shown when only GOTURN is correct. The red, green, white and purple boxes correspond to the outputs of TLD, CMT, STRUCK and GOTURN respectively. The yellow box is the output of the fused approach. Best viewed in color.

We observe that the primary benefit of our method is in challenging scenarios such as when a tracker is lost for many frames. Going back to our example of the *bolt* sequence as shown in Fig. 5.9, we saw that in our prior algorithm, the tracker would be confused by the two clusters of STRUCK/CMT and GOTURN/TLD. We see in Fig. 5.16 that our method is able to keep track of the runner for much longer despite the fact that the performance of the trackers remained the same. Whereas the previous approach loses track of the target at frame 20, our proposed method is able to track it accurately until frame 100. Naturally, the algorithm would move towards STRUCK as this was the most trusted tracker and had been following that trajectory for a long time. But by using our autoencoder approach, we believe our network is able to statistically learn that a target in the corner is not as likely and therefore we can penalize that tracker accordingly. We also see in Fig. 5.15, that our method is now able to handle some more scenarios that used to pose a challenge. In particular, this result is important because it shows the benefit of including generally weaker trackers but that rely on alternative visual features than those used by the better performing trackers.

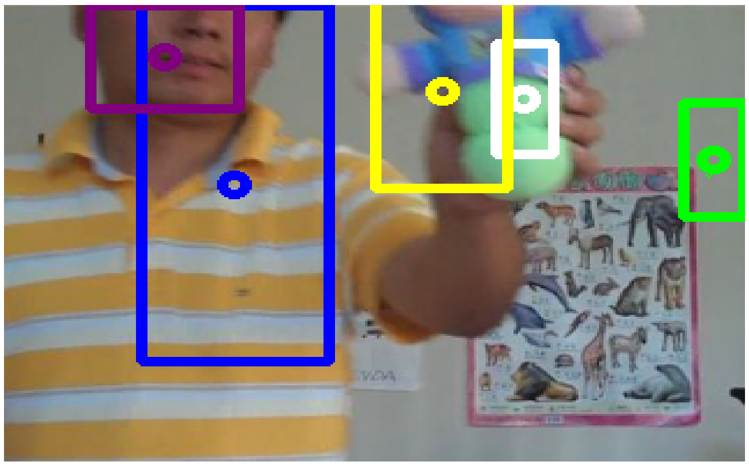


Figure 5.17: Robustness to failure from all but one of the trackers. We see that STRUCK is correctly chosen to be trusted here. See the caption of Fig. 5.16 for a description of the elements of the figure. Best viewed in color.

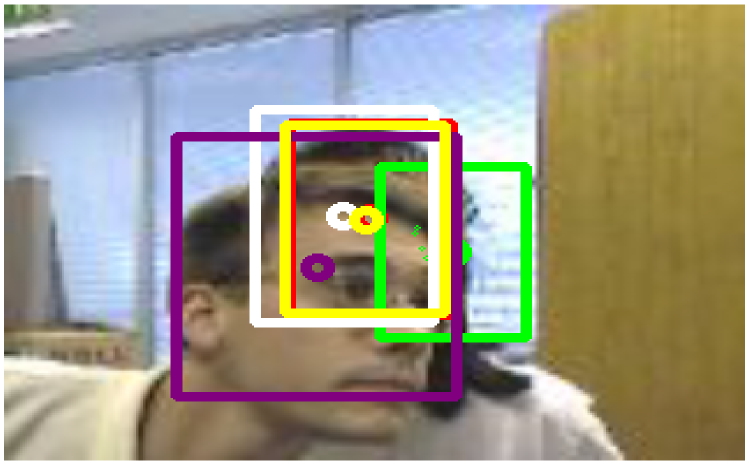


Figure 5.18: Robustness to the failure of multiple trackers. Here, we also observe that the algorithm correctly chooses to trust TLD more. See the caption of Fig. 5.16 for a description of the elements of the figure. Best viewed in color.

We notice that our ensemble produces benefits not only in terms of increased success (i.e., having overlap with the ground truth), but also in increased precision (having a better overlap) as well. This is especially evident in Fig. 5.18. This is because the algorithm can select which tracker to trust more, by properly doing so, the results obtained are much more accurate. The anomaly detection design is also shown to be beneficial. If we phrase the problem as an outlier detection problem, situations such as those in Fig. 5.17 become easier to tackle in future research. All that is required is to detect that the trackers are failing, while we explore autoencoders, handcrafted features and other methods can all be integrated in this method. Rather than using just the autoencoder, the maximum a posteriori can additionally incorporate probabilities from other classifiers. However, there are still issues with the approach. There are instances when the algorithm incorrectly highly trusts a tracker. This tends to confuse the algorithm and degrade results. However, these are problems associated with the network, with better training, this problem can be tackled and mitigated.

5.4.2 Quantitative Results on Training Set

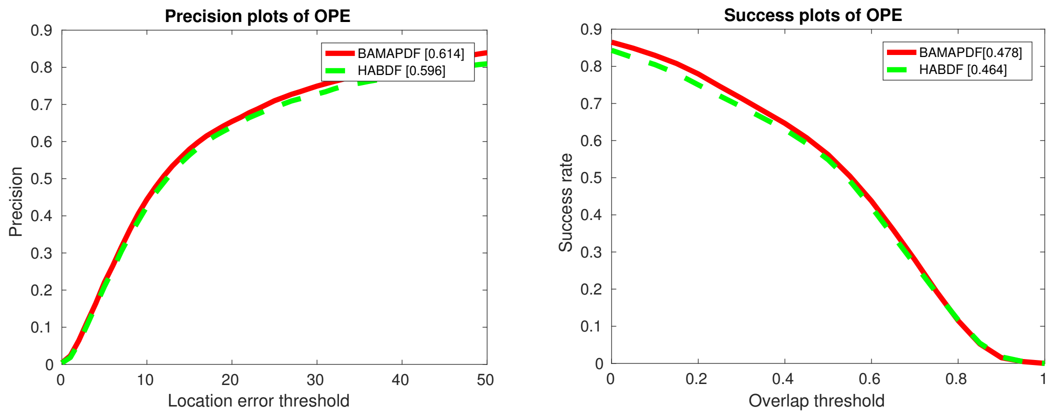


Figure 5.19: Results of our Tracker (BAMAPDF) on OPE for OTB-50 compared to the original implementation.

By running our tracker ensemble on the OTB-50 dataset we compare the quantitative benefits of our network on the benchmark. We observe an improvement of approximately 3% on both precision and success as seen in Fig. 5.19. While this may not seem very high, we would like to point out that the original Hierarchical Bayesian Data Fusion was already a 5% increase over the best tracker in the ensemble. The increase in certain key scenarios was substantial and goes a great deal in mitigating a large weakness of the original implementation.

Our largest improvement came in the low resolution sequences, where we observed a 27% increase in precision and a 26% increase in success rate as shown in Fig. 5.20. While this still does not compensate for the drastic 60% worse performance of the original algorithm when compared to the best tracker in the ensemble, it is still a significant improvement.

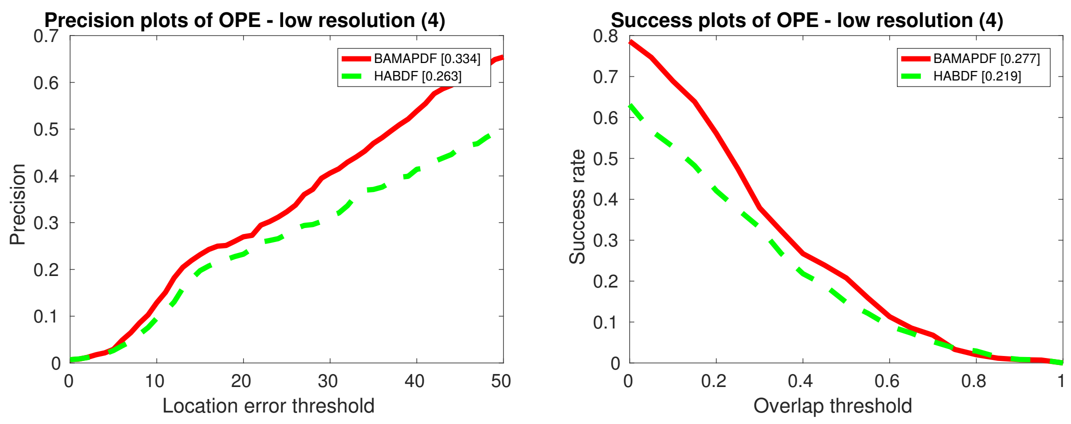


Figure 5.20: Results of our Tracker (BAMAPDF) on OPE for the Low Resolution case, here we see the largest improvement relative to the initial implementation.

We also observe that in most cases, the new approach provides better or comparable results. In Fig. 5.21, which presents the results for the occlusion scenario, we notice the least improvement, but still see that our results are 1%

better. We provide the complete results in Table. 5.1

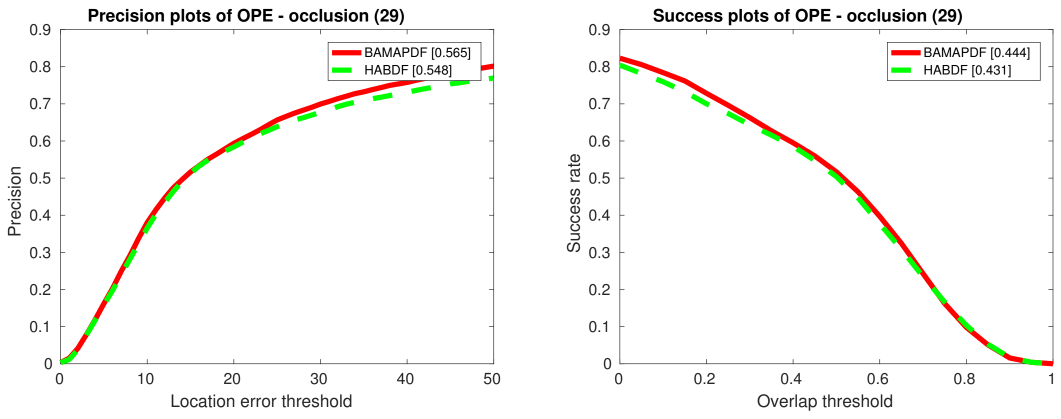


Figure 5.21: Results of our Tracker (BAMAPDF) on occlusion, here the improvement is the smallest, but small incremental improvement is seen.

5.4.3 Summary of Results

In this chapter, we introduced Bayesian Autoencoder Maximum Likelihood Data Fusion (BAMAPDF). We substituted the original Mahalanobis distance metric with an autoencoder based maximum a posteriori. We used this likelihood to compute a score that can be used by our modified algorithm. We evaluated this modified algorithm on a simulated data problem and finally evaluated our updated algorithm on a benchmark. Our approach provided better results as measured by the benchmark. Although our approach could be further improved, our improvements on the OTB have proven that it is a successful paradigm worthy of further exploration with many available avenues of improvement. We believe this is a promising algorithm that can be integrated into a UAV to follow an object.

Table 5.1: Summary of Results on OPE: Training Set

Scenario	BAMAPDF Score (Precision /Success)	HABDF Score (Precision /Success)	Percent Change Relative to Best Tracker
Total	.614/.478	.596/.464	+3.020% / +3.017%
illumination	.558/.461	.524/.427	+5.681% / +7.683%
out-of -plane rotation	.589/.460	.575/.448	+2.434%/+2.679%
scale variation	.601/.455	.574/.432	+4.704%/+5.324%
occlusion	.565/.444	.548/.431	+3.102% / +3.102%
deformation	.574/.453	.568/.450	+1.056%/+0.667%
motion blur	.489/.408	.450/.366	+8.667% /+11.475%
fast motion	.493/.413	.455/.377	+8.351%/+9.495%
in-plane rotation	.588/.462	.556/.439	+5.755% / +5.239%
out of view	.500/.479	.465/.441	+7.529% /+8.617%
background clutter	.570/.458	.547/.437	+4.204% / +4.805%
low resolution	.334/.277	.263/.219	+27.000%/+26.848%

CHAPTER 6

CONCLUSION

Our goal was to contribute to this field of “robust vision-based target tracking” to eventually be used in an autonomous follow me UAV. In this thesis, we propose an improvement to the Hierarchical Bayesian Data Fusion algorithm. Specifically, we proposed improving the current algorithm by using a maximum a posteriori approach that uses autoencoders. First, we examined the original method quantitatively using the OTB-50 benchmark dataset. To address issues with the Mahalanobis distance score, we specifically explored the use of autoencoders as sources of information. In particular, we used the reconstruction error of the autoencoder to detect whether a frame is an outlier or not.

To train our autoencoder, we first proposed the method that could detect that any tracker was an outlier and we evaluated various network architectures. Once we saw that this formulation was insufficient for our application, we proposed a different paradigm that focused on the probability of abnormality of our trackers. We evaluated our final architecture and assessed the success rate with which it detect anomalies. Next, we used Bayesian statistics to transform our classifier into a score metric.

After performing simulations to determine that our metric was reasonable we substituted the Mahalanobis metric in the original work with our proposed approach. We ran our updated approach on the benchmark to compare our new to the original implementation for a vis-a-vis comparison. We saw that our method provided improvement to the original approach and motivates further research in this area.

6.1 Contribution

To this end, our three main contributions include:

1. Detecting anomalies when only one tracker is lost;
2. Detecting anomalies when one tracker is lost and we do not care about the other trackers;
3. Integrating our autoencoder-based confidence score algorithm into a tracker ensemble.

We observed that the autoencoder was capable of detecting when only one tracker was lost better than the Mahalanobis distance approach. When one tracker is lost and we do not care about the other trackers, anomaly detection proved to be a more challenging problem. However, we were able to mitigate this problem by compensating for the zeroth degree moment by applying a moving offset. One important thing to note is that although we got the performance of the autoencoders to their current level in our research, our system is modular so that alternative networks could be substituted easily into our approach.

Finally, a different histogram method that accounts for this offset would also be beneficial. Lastly, despite all the areas of possible improvement, we saw that our method was able to perform better on the benchmark dataset than the baseline.

6.2 Future Work

Our immediate future goal is to integrate this method with a pan and tilt camera and UAV and evaluate the performance in the wild. Our approach would be heavily inspired by Echeverri et al.'s work in [18]. However, our pan and tilt used would be the DJI Zenmuse X3 which would act in a similar manner to a pan and tilt and would likely only require the PID control values to change. Since the

Zenmuse can easily be attached to a DJI Matrice drone, the next step would be to fly the drone to autonomously follow an object. In both of these cases, our Bayesian Autoencoder Maximum A Posteriori Data Fusion (BAMAPDF) algorithm would be ported onto a laptop used to run the code. Using ROS and the DJI ROS onboard SDK, flight commands would be sent via a WLAN, bluetooth or WiFi connection to a Raspberry Pi computer running ROS, which would directly send commands via UART to the drone. The drone would act as a listener and execute these commands. To acquire near real-time images, we would stream directly from DJI flight controllers mini HDMI port. We show this goal and future implementation in Fig. 6.1.

With more data and a richer model, improvement is definitely possible. In fact, richer features would very likely lead to improvement. In particular, using visual features would be a particular route that could be accommodated into the framework. Specifically, correlation filter based approaches such as the work done by Walsh [83] have proven that anomaly detection using correlation filters is possible. Additionally many current trackers, such as DCPF [60, 30], use correlation maps with high levels of success and these maps could be used as features. Additionally, it is also possible that there are better ways to compute the offset. By performing a statistical analysis of the reconstruction errors, a much better offset rule could be determined which would likely yield improved results.

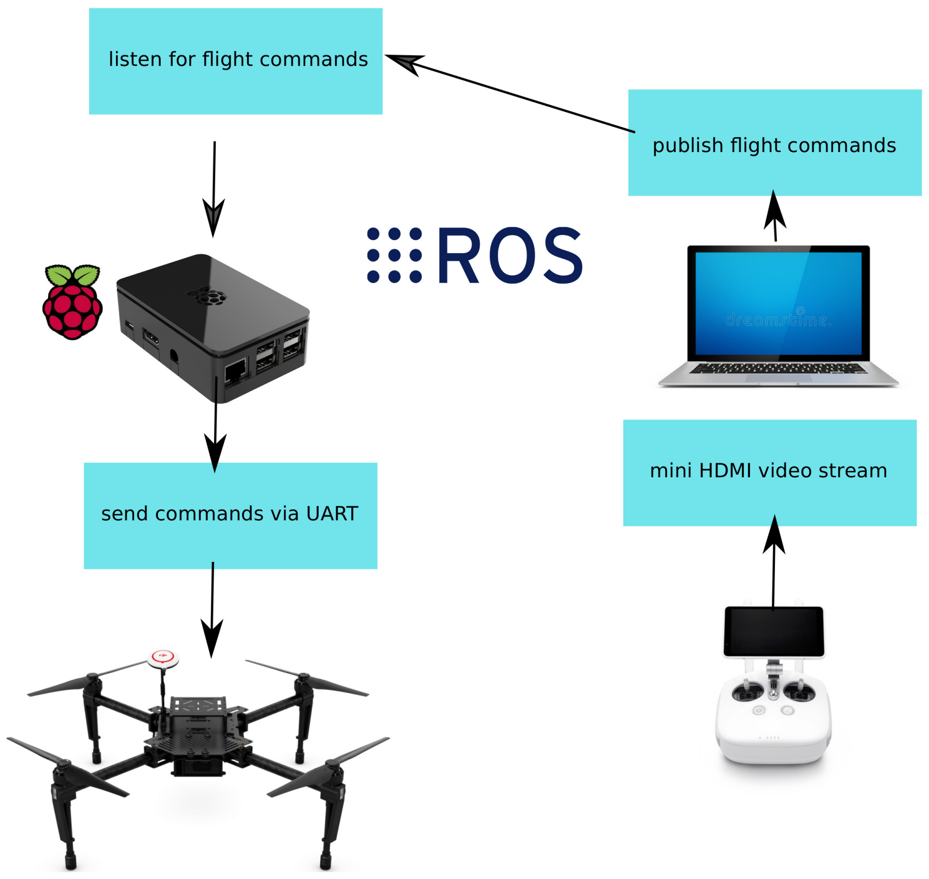


Figure 6.1: Schematic representation of our proposed final goal. Best viewed in color.

BIBLIOGRAPHY

[1] Hamid Alipour, Daniel Zeng, and Douglas C Derrick. Adaboost-based sensor fusion for credibility assessment. pages 224–226. IEEE, 2012.

[2] Nahla Ben Amor, Salem Benferhat, and Zied Elouedi. Naive bayes vs decision trees in intrusion detection systems. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 420–424. ACM, 2004.

[3] Christian Bailer, Alain Pagani, and Didier Stricker. *A Superior Tracking Approach: Building a Strong Tracker through Fusion*, pages 170–185. Springer International Publishing, Cham, 2014.

[4] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016.

[5] Keshav Bimbraw. Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. In *Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on*, volume 1, pages 191–198. IEEE, 2015.

[6] Tewodros Atanaw Biresaw, Andrea Cavallaro, and Carlo S. Regazzoni. Tracker-level fusion for robust bayesian visual tracking. *IEEE Trans. Circuits Syst. Video Techn.*, 25(5):776–789, 2015.

[7] Cara Bloom and Joshua Tan Javed Ramjohn Lujo Bauer. Self-driving cars and data collection: Privacy perceptions of networked autonomous vehicles. In *Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[8] Erkan Bostanci, Betul Bostanci, Nadia Kanwal, and Adrian F Clark. Sensor fusion of camera, gps and imu using fuzzy adaptive multiple motion models. *Soft Computing*, 22(8):2619–2632, 2018.

[9] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[10] Robert Grover Brown, Patrick YC Hwang, et al. *Introduction to random signals and applied Kalman filtering*, volume 3. Wiley New York, 1992.

[11] Wassim S. Chaer, Robert H. Bishop, and Joydeep Ghosh. A mixture-of-experts framework for adaptive kalman filtering. *IEEE Trans.*

Systems, Man, and Cybernetics, Part B, 27(3):452–464, 1997.

[12] R. Chalapathy, A. Krishna Menon, and S. Chawla. Anomaly Detection using One-Class Neural Networks. *ArXiv e-prints*, February 2018.

[13] L. Chin. Application of neural networks in target tracking data fusion. *IEEE Transactions on Aerospace and Electronic Systems*, 30(1):281–287, Jan 1994.

[14] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[15] Philipe A Dias, Amy Tabb, and Henry Medeiros. Apple flower detection using deep convolutional networks. *Computers in Industry*, 99:17–28, 2018.

[16] Miaobo Dong, Ben M Chen, Guowei Cai, and Kemao Peng. Development of a real-time onboard and ground station software system for a uav helicopter. *Journal of Aerospace Computing, Information, and Communication*, 4(8):933–955, 2007.

[17] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: fifteen years later. 2011.

[18] Andres F. Echeverri, Henry Medeiros, Ryan Walsh, Yevgeniy Reznichenko, and Richard J. Povinelli. Hierarchical bayesian data fusion for robotic platform navigation. *ICUAS*, 2018.

[19] Mica R Endsley. Autonomous driving systems: A preliminary naturalistic study of the tesla model s. *Journal of Cognitive Engineering and Decision Making*, 11(3):225–238, 2017.

[20] H. Fan and H. Ling. Sanet: Structure-aware network for visual tracking. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2217–2224, July 2017.

[21] Mário AT Figueiredo. Adaptive sparseness for supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 25(9):1150–1159, 2003.

[22] Ben Goertzel, Julia Mossbridge, Eddie Monroe, David T Hanson, and Gino Yu. Humanoid robots as agents of human consciousness expansion. *CoRR*, abs/1709.07791, 2017.

[23] Mahanth Gowda, Justin Manweiler, Romit Roy Choudhury, and Justin D Weisz. Tracking Drone Orientation with Multiple GPS Receivers. 2016.

[24] Yu Gu, Jason N. Gross, Matthew B. Rhudy, and Kyle Lassak. A Fault-Tolerant Multiple Sensor Fusion Approach Applied to UAV Attitude Estimation. *International Journal of Aerospace Engineering*, 2016, 2016.

[25] Sam Hare, Amir Saffari, and Philip H. S. Torr. Struck: Structured output tracking with kernels. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 263–270. IEEE Computer Society, 2011.

[26] 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.

[27] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference Computer Vision (ECCV)*, 2016.

[28] Tsai Hong Hong, Christopher Rasmussen, Tommy Chang, and Michael Shneier. Road detection and tracking for autonomous mobile robots. In *Unmanned Ground Vehicle Technology IV*, volume 4715, pages 311–320. International Society for Optics and Photonics, 2002.

[29] Weiming Hu, Wei Hu, and Steve Maybank. Adaboost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(2):577–583, 2008.

[30] Reza Jalil Mozhdehi and Henry Medeiros. Deep convolutional particle filter for visual tracking. In *2017 IEEE International Conference on Image Processing (ICIP2017)*, 2017.

[31] Reza Jalil Mozhdehi, Yevgeniy Reznichenko, Abubakar Siddique, and Henry Medeiros. Deep convolutional particle filter with adaptive correlation maps for visual tracking. *2018 IEEE International Conference on Image Processing (ICIP2018)*, 2018.

[32] Vijay Manikandan Janakiraman and David Nielsen. Anomaly detection in aviation data using extreme learning machines. pages 1993–2000. IEEE, 2016.

[33] G. Jiang, P. Xie, H. He, and J. Yan. Wind turbine fault detection using denoising autoencoder with temporal information. *IEEE/ASME Transactions on Mechatronics*, PP(99):1–1, 2017.

[34] Sunghun Jung and Hyunsu Kim. Analysis of amazon prime air uav delivery service. *Journal of Knowledge Information Technology and Systems*, 12:253–266,

04 2017.

[35] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1409–1422, July 2012.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[37] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *stat*, 1050:1, 2014.

[38] S. Kiranyaz, T. Ince, and M. Gabbouj. Real-time patient-specific ecg classification by 1-d convolutional neural networks. *IEEE Transactions on Biomedical Engineering*, 63(3):664–675, March 2016.

[39] P Koprinkova and M Petrova. Data-scaling problems in neural-network training. *Engineering Applications of Artificial Intelligence*, 12(3):281–296, 1999.

[40] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *stat*, 1050:16, 2015.

[41] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman P. Pflugfelder, and Luka et al. Cehovin. The visual object tracking vot2016 challenge results. In Gang Hua and Herv Jgou, editors, *ECCV Workshops (2)*, volume 9914 of *Lecture Notes in Computer Science*, pages 777–823, 2016.

[42] Alex Krizhevsky and Geoffrey E. Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[45] L. I. Kuncheva and J. J. Rodríguez. A weighted voting framework for classifiers ensembles. *Knowledge and Information Systems*, 38(2):259–275, 2014.

[46] Ido Leichter, Michael Lindenbaum, and Ehud Rivlin. A general framework for combining visual trackers–the" black boxes" approach. *International Journal of Computer Vision*, 67(3):343–363, 2006.

[47] Pengpeng Liang, Erik Blasch, and Haibin Ling. Encoding color information for visual tracking: Algorithms and benchmark. *IEEE Transactions on Image Processing*, 24(12):5630–5644, 2015.

[48] Shaopeng Liu, Robert X Gao, Dinesh John, John Staudenmayer, and Patty S Freedson. Svm-based multi-sensor fusion for free-living physical activity assessment. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pages 3188–3191. IEEE, 2011.

[49] Yan-Jun Liu, Li Tang, Shaocheng Tong, CL Philip Chen, and Dong-Juan Li. Reinforcement learning design-based adaptive tracking control with less learning parameters for nonlinear discrete-time mimo systems. *IEEE Transactions on Neural Networks and Learning Systems*, 26(1):165–176, 2015.

[50] Manuel Lopez-Martin, Belén Carro, Antonio Sánchez-Esguevillas, and J. Rodríguez Lloret. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot. In *Sensors*, 2017.

[51] Gareth Loy, Luke Fletcher, Nicholas Apostoloff, and Alexander Zelinsky. An adaptive fusion architecture for target tracking. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 261–266. IEEE, 2002.

[52] Yunlong Ma, Peng Zhang, Yanan Cao, and Li Guo. Parallel auto-encoder for efficient outlier detection. *Proceedings - 2013 IEEE International Conference on Big Data, Big Data 2013*, 2(3):15–17, 2013.

[53] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[54] Wenguang Mao and Lili Qiu. Dronetrack: An indoor follow-me system using acoustic signals. *GetMobile: Mobile Computing and Communications*, 21(4):22–24, 2018.

[55] Erik Marchi, Fabio Vesperini, Florian Eyben, Stefano Squartini, and Björn Schuller. A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional lstm neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1996–2000. IEEE, 2015.

[56] Erik Marchi, Fabio Vesperini, Stefano Squartini, and Björn Schuller. Deep recurrent neural network-based autoencoders for acoustic novelty detection. *Computational intelligence and neuroscience*, 2017, 2017.

[57] Markos Markou and Sameer Singh. Novelty detection: A review - Part 2:: Neural network based approaches. *Signal Processing*, 83(12):2499–2521, 2003.

[58] Kourosh Meshgi, Maryam Sadat Mirzaei, Shigeyuki Oba, and Shin Ishii. Active collaborative ensemble tracking. *CoRR*, abs/1704.08821, 2017.

[59] Emilie Morvant, Amaury Habrard, and Stéphane Ayache. Majority vote of diverse classifiers for late fusion. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 153–162. Springer, 2014.

[60] R. J. Mozhdehi and H. Medeiros. Deep convolutional particle filter for visual tracking. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3650–3654, Sept 2017.

[61] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 4293–4302. IEEE, 2016.

[62] Georg Nebehay and Roman Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision*, pages 862–869. IEEE, March 2014.

[63] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[64] Roberto Perdisci, Guofei Gu, and Wenke Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 488–498. IEEE, 2006.

[65] Daniel Perez, Ivan Maza, Fernando Caballero, David Scarlatti, Enrique Casado, and Anibal Ollero. A ground control station for a multi-uav surveillance system. *Journal of Intelligent & Robotic Systems*, 69(1-4):119–130, 2013.

[66] Jesus Pestana, Jose Luis Sanchez-Lopez, Srikanth Saripalli, and Pascual Campoy. Computer vision based general object following for gps-denied multirotor unmanned vehicles. In *American Control Conference (ACC), 2014*, pages 1886–1891. IEEE, 2014.

[67] Raquel Ramos Pinho, João Manuel R. S. Tavares, and Miguel V. Correia. Efficient approximation of the mahalanobis distance for tracking with the

kalman filter. In João Manuel R. S. Tavares and Renato M. Natal Jorge, editors, *Computational Modeling of Objects Represented in Images-Fundamentals, Methods and Applications, First International Symposium CompIMAGE 2006, Coimbra, Portugal, October 20-21, 2006.*, pages 349–354. Taylor & Francis, 2006.

[68] Yevgeniy Reznichenko and Henry Medeiros. Improving target tracking robustness with bayesian data fusion. In *British Machine Vision Conference*, September 2017.

[69] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):109–120, 2007.

[70] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[71] Inkyu Sa, Mina Kamel, Raghav Khanna, Marija Popovic, Juan Nieto, and Roland Siegwart. Dynamic system identification, and control for a cost effective open-source vtol mav. *arXiv preprint arXiv:1701.08623*, 2017.

[72] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

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

[74] Vasilios A Siris and Fotini Papagalou. Application of anomaly detection algorithms for detecting syn flooding attacks. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 4, pages 2050–2054. IEEE.

[75] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, 2014.

[76] Halil Ersin Soken and Chingiz Hajiyev. Adaptive unscented kalman filter with multiple fading factors for pico satellite attitude estimation. In *Recent Advances in Space Technologies, 2009. RAST'09. 4th International Conference on*, pages 541–546. IEEE, 2009.

[77] Veronika Stefanov and Beate List. Outlier Detection Using Replicator Neural Networks. *Data Warehousing and Knowledge Discovery*, 4654(DECEMBER):209–220, 2007.

[78] Gary Stein, Bing Chen, Annie S Wu, and Kien A Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *Proceedings of the 43rd annual Southeast regional conference-Volume 2*, pages 136–141. ACM, 2005.

[79] HTM Tran and DC Hogg. Anomaly detection using a convolutional winner-take-all autoencoder. In *British Machine Vision Conference*, September 2017.

[80] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125. ISCA, 2016.

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[82] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[83] Ryan Walsh and Henry Medeiros. Detecting tracking failures from correlation response maps. In *International Symposium on Visual Computing*, pages 125–135. Springer, 2016.

[84] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[85] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *CVPR*, pages 2411–2418. IEEE Computer Society, 2013.

[86] Mau-Tsuen Yang and Shen-Yen Huang. Appearance-based multimodal human tracking and identification for healthcare in the digital home. *Sensors*, 14(8):14253–14277, 2014.

[87] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. Twenty years of mixture of experts. *IEEE Trans. Neural Netw. Learning Syst.*, 23(8):1177–1193, 2012.

[88] Jianming Zhang, Shugao Ma, and Stan Sclaroff. Meem: robust tracking via multiple experts using entropy minimization. In *European Conference on Computer Vision*, pages 188–203. Springer, 2014.

[89] Wei Zhang, Gaoliang Peng, Chuanhao Li, Yuanhang Chen, and Zhujun Zhang. A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals. In *Sensors*, 2017.

[90] Chong Zhou and Randy C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 665–674, New York, NY, USA, 2017. ACM.