# END-TO-END NETWORKS FOR DETECTION AND TRACKING OF MICRO UNMANNED AERIAL VEHICLES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CEMAL AKER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2018

Approval of the thesis:

**END-TO-END NETWORKS FOR DETECTION AND TRACKING OF MICRO UNMANNED AERIAL VEHICLES**

submitted by **CEMAL AKER** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halit Oğuztüzün _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Sinan Kalkan _____
Supervisor, **Computer Engineering Department, METU**

**Examining Committee Members:**

Assist. Prof. Dr. Ramazan Gökberk Cinbiş
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Erkut Erdem
Computer Engineering Department, Hacettepe University _____

**Date:** _____

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last Name: CEMAL AKER

Signature :

# ABSTRACT

## END-TO-END NETWORKS FOR DETECTION AND TRACKING OF MICRO UNMANNED AERIAL VEHICLES

Aker, Cemal
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Sinan Kalkan

September 2018, 64 pages

As the number of micro unmanned aerial vehicles (mUAV) increases, several threats arise. Hence, there is a need for a system that can detect and track them. In this thesis, an object detection model based on convolutional neural networks for mUAV detection, and a novel end-to-end object tracking architecture are proposed. To solve the scarce data problem for training the detection network, an algorithm for creating an extensive artificial dataset by combining background-subtracted real images is proposed. It has been shown that the created dataset is adequate for training well performing networks and that the system can detect and track various types of mUAVs in challenging environments.

Keywords: Object Detection, Object Tracking, Convolutional Neural Networks, Neural Turing Machine, Deep Learning

# ÖZ

## MİKRO İNSANSIZ HAVA ARAÇLARININ TESPİTİ VE TAKİBİ İÇİN UÇTAN UCA AĞLAR

Aker, Cemal
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Sinan Kalkan

Eylül 2018, 64 sayfa

Mikro insansız hava araçlarının (mİHA) sayısı arttıkça, çok çeşitli tehditler gündeme gelmektedir. Bu yüzden, bu araçları tespit ve takit edebilen sistemlere ihtiyaç duyulmaktadır. Bu tezde, mİHA tespiti için evrişimsel sinir ağı tabanlı uçtan uca nesne tespit modeli ile birlikte, özgün bir nesne takip mimarisi önerilmiştir. Tespit ağını eğitmek için yetersiz veri problemini çözmek adına, arka planı temizlenmiş gerçek imgeler kullanan geniş çaplı veri kümesi oluşturma yöntemi önerilmiştir. Oluşturulan veri kümesinin, iyi çalışan sinir ağları eğitmek için yeterli olduğu ve sistemin zorlayıcı ortamlarda çok çeşitli mİHA türlerini tespit ve takip edebileceği gösterilmiştir.

Anahtar Kelimeler: Nesne Tespiti, Nesne Takibi, Evrişimsel Sinir Ağları, Nöral Turing Makinesi, Derin Öğrenme

To my family and beloved ones.

## ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Assoc. Prof. Dr. Sinan Kalkan for his guidance, support, friendship, encouragements and toleration throughout the preparation of this thesis.

I would like to thank Hande Çelikkanat for her kind friendship, encouragements and help. She was the one who hold my hands when I got stuck.

I would like to thank my labmates Irmak Doğan, Osman Tursun, Ezgi Ekiz, İlker Bozcan, Negin Bagherzadi, Yunus Terzioğlu, Sera Büyükgöz and Fatih Can Kurnaz.

I would like to thank NVIDIA Corporation for the hardware support. I would also like to thank Turkish Scientific and Technological Research Council (TÜBİTAK) for their financial support during my M.Sc. studies.

Last but not least, I would like to thank my family and beloved ones for being always with me and their support originating from their invaluable hearts.

## TABLE OF CONTENTS

## LIST OF TABLES

TABLES

# LIST OF FIGURES

FIGURES

# LIST OF ALGORITHMS

ALGORITHMS

## LIST OF ABBREVIATIONS

| | |
|---|---|
| mUAV | Micro Aerial Vehicle |
| CNN | Convolutional Neural Network |
| RNN | Recurrent Neural Network |
| LSTM | Long Short Term Memory |
| NTM | Neural Turing Machine |
| NTT | Neural Turing Tracker |
| YOLO | You Only Look Once |
| YOLOv2 | You Only Look Once Version 2 |
| IOU | Intersersection Over Union |
| NIN | Network In Network |
| BOVW | Bag of Visual Words |
| FAST | FAST Corner Detection |
| HARRIS | Harris Corner Detection |
| GFTT | Good Features To Track |
| MSER | Maximally Stable Extremal Region extractor |
| SIFT | Scale Invariant Feature Transform |
| SURF | Speeded-up Robust Features |
| HOG | Histogram of Oriented Gradients |
| BRIEF | Binary Robust Independent Elementary Features |
| ORB | Oriented FAST and Rotated BRIEF |
| SVM | Support Vector Machine |
| SSD | Single Shot Multibox Detector |
| R-CNN | Regions with CNN |
| SPP | Spatial Pyramid Pooling |
| ROI | Region of Interest |
| RPN | Region Proposal Network |
| IVT | Incremental Visual Tracking |

| | |
|---|---|
| MIL | Multiple Instance Learning |
| MOSSE | Minimum Output Sum of Squared Error |
| SRDCF | Spatially Regularized Discriminative Correlation Filters |
| TLD | Tracking Learning Detection |
| BPTT | Back Propagation Through Time |
| D-NTM | Dynamic Neural Turing Machines |
| PR | Precision-Recall |

# CHAPTER 1

# INTRODUCTION

The technology needed to design and manufacture micro unmanned aerial vehicles (mUAV)[1] has advanced rapidly in recent years. This allowed corporations to mass produce drones[2]. In these days, everyone can own a drone with a high definition camera attached to it, for very low prices, even from super markets. The availability of drones make it threatening since they can harm the privacy of people by taking photos or recording videos unbeknown to them. Similarly, they can be used to gather visual and audio information from places where security is crucial. In addition to these general purpose drones, armed forces and defense industry are interested in armed ones, which also motivates for examining them for security applications.

Increasing number of drones in the sky introduces new problems for pilots. Since they are small in size, the pilots of manned air vehicles may fail to notice, which may result in collisions and loss of lives and property. On the other side, one may want to pilot many drones at the same time, i.e., drone swarm. The rise in popularity creates a demand for solution of such problems.

With the mentioned motivation, it is essential to detect drones before any undesired circumstances, track them until they disappear and act for defense if possible. Likewise, it essential for them to detect and track others to form a group.

The rapid improvement in machine learning, especially deep learning, area has recently enabled most of the computer vision problems to be almost solved. With this

[1] mUAVs are unmanned aerial vehicles less than 5kg.
[2] mUAVs are often called drones. This name is used especially for multi-copters.

light, the mentioned problems are attacked as end-to-end learning problems in this thesis.

## 1.1 Problem Definition

The following problems are addressed in this thesis:

- **Detecting drones:** Given an image, the aim is to report pixel coordinates of the centers of the drones in the image along with its extents. This is basically visual object detection adapted specifically to drones. However, they resemble birds in the sky a lot. Hence the detection method should be able to distinguish between them.
- **Tracking drones:** Given a sequence of video frames along with drone detection result for each time step, the goal is to refine detection so that the noise in detection process is eliminated, the occlusion of the main object is handled, and its behavior is estimated to be able to make further refinements.

## 1.2 Contributions

The main contributions of the thesis are as follows:

- **Investigation of a deep learning based method on drone detection task:** In this thesis, it has been shown that a deep convolutional neural network based object detection network can be trained to detect drones under harsh conditions such as bad illumination, small scale and clutter. A thorough search of the relevant literature had yielded no published deep learning based method for visual drone detection problem until the Drone-vs-Bird Detection Challenge[3][4] was held. Although the proposed method had the third best results, it has shown

[3] Challenge was held in conjunction with the "International workshop on small-drone surveillance, detection and counteraction techniques" of IEEE AVSS 2017.

[4] https://wosdetc.wordpress.com/challenge/

along with the studies ranking highest in the challenge that deep learning has a very crucial role in drone detection problem.

This part of the thesis is published within the scope of the challenge [1], [17].

- **METU Drone Dataset:** Since flying time of the drones are limited, and recording and labeling hours of videos for deep learning frameworks is very tremendous, a novel artificial dataset creation algorithm is proposed. The created drone detection dataset is published online.

- **A novel end-to-end network for object tracking:** Considering the results, which state that detection is not adequate on its own, of aforementioned challenge, a novel object tracking approach is proposed to take the time domain information into consideration.

The contributions presented in this thesis are disseminated in the following studies:

- **Cemal Aker** and Sinan Kalkan. Using Deep Networks for Drone Detection. 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2017.

- Angelo Coluccia, Marian Ghenescu, Tomas Piatrik, Geert De Cubber, Arne Schumann, Lars Sommer, Johannes Klatte, Tobias Schuchert, Juergen Beyerer, Mohammad Farhadi, Ruhallah Amandi, **Cemal Aker**, Sinan Kalkan, Muhammad Saqib, Nabin Sharma, Sultan Daud, Michael Blumenstein. Drone-vs-Bird detection challenge at IEEE AVSS2017. 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2017.

## 1.3 Organization

In Chapter 2, the related literature is reviewed by focusing on general purpose methods for object detection and tracking, and on specifically drone detection and tracking. Moreover, background information about recurrent neural networks, their variants and Neural Turing Machine (NTM) architectures is provided.

In Chapter 3, an end-to-end deep convolutional network based mUAV detection and localization method is proposed. The chapter includes details of both network architecture and training details. This part of the thesis has been published in [1].

In Chapter 4, a novel object tracking method, that is based on NTM, is proposed. The chapter describes the modules of the model, data flow and training details.

In Chapter 5, the details of the datasets and proposed artificial dataset creation method are described. The chapter also describes the evaluation metrics and experimental results for drone detection problem.

In Chapter 6, closing remarks for the thesis are provided by summarizing the proposed methods, their limitations and pathways for further studies.

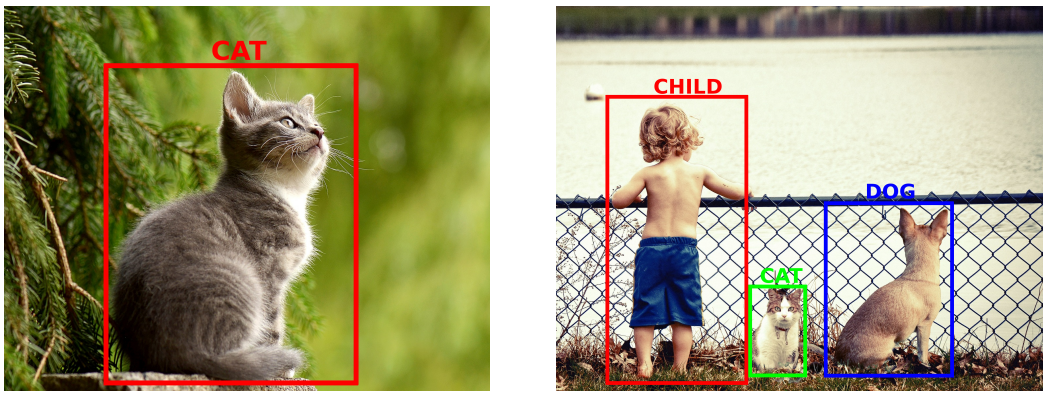
# CHAPTER 2

# RELATED WORK AND BACKGROUND

In this chapter, object detection and tracking methods, and using them specific to drones are reviewed. In addition, background information about recurrent neural networks, its variants and neural turing machines is provided.

## 2.1 Object Detection and Localization

Object localization is the problem of determining a bounding box and class for an image that includes only one object of interest. However, object detection requires to detect all instances of all relevant categories in an image.




(a) Object localization (b) Object detection

Figure 2.1: Examples for object localization and object detection problems.

### 2.1.1 Classical Methods

The widely used classical methods are all based on a technique called sliding window. In this technique, a window is cropped from each position of a query image, and a classifier is used to determine whether the window has an object of interest or background. Figure 2.2 shows how sliding window technique is applied. This approach fails to detect objects when their scale changes. In order to overcome this, the technique is applied to an image pyramid of the query which consists of different scales of the same image. If multiple detections are done for a single object, non-maximum suppression is applied to choose the one with the highest response.

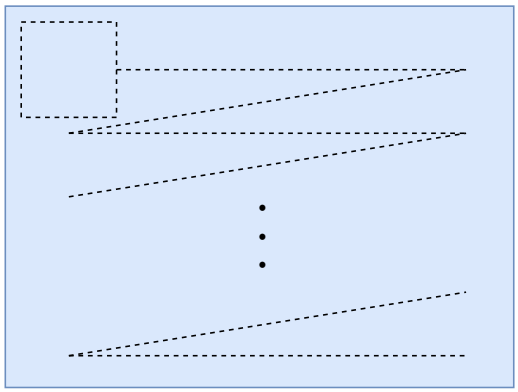


Figure 2.2: Sliding window technique

In order to classify a given image patch, one needs a means of gathering high level information from the pixels and a strong classifier that discriminates between the objects and background using that information. The most widely used methods are based on bag of visual words (BOVW) [99], or Viola Jones framework [105].

In BOVW approach, the first step is detecting keypoints that are assumed to be important to describe objects with the help of keypoint detectors such as Fast corner detection (FAST) [103], Harris corner detection (HARRIS) [42], Good Features To Track (GFTT) [95] and Maximally Stable Extremal Region extractor (MSER) [67]. In order to extract mid-level information, the subwindows surrounding the keypoints are described by local feature descriptors such as Scale Invariant Feature Transform (SIFT) [63], Speeded-up Robust Features (SURF) [5], Histogram of Oriented Gra-

dients (HOG) [22], Binary Robust Independent Elementary Features (BRIEF) [14], and Oriented FAST and Rotated BRIEF (ORB) [88]. The resulting feature vectors from all patches including objects or background are quantized using a clustering algorithm (e.g., K-means clustering [62]) to form a dictionary of visual words. While forming the high level features for image patches, keypoints are detected, described, and an histogram of corresponding visual words are formed. The feature vectors representing the high level information in image patches are used as data instance for a discriminative machine learning model; e.g., support vector machines (SVM) [20], in training and testing stages. The overall approach is described in Figure 2.3.

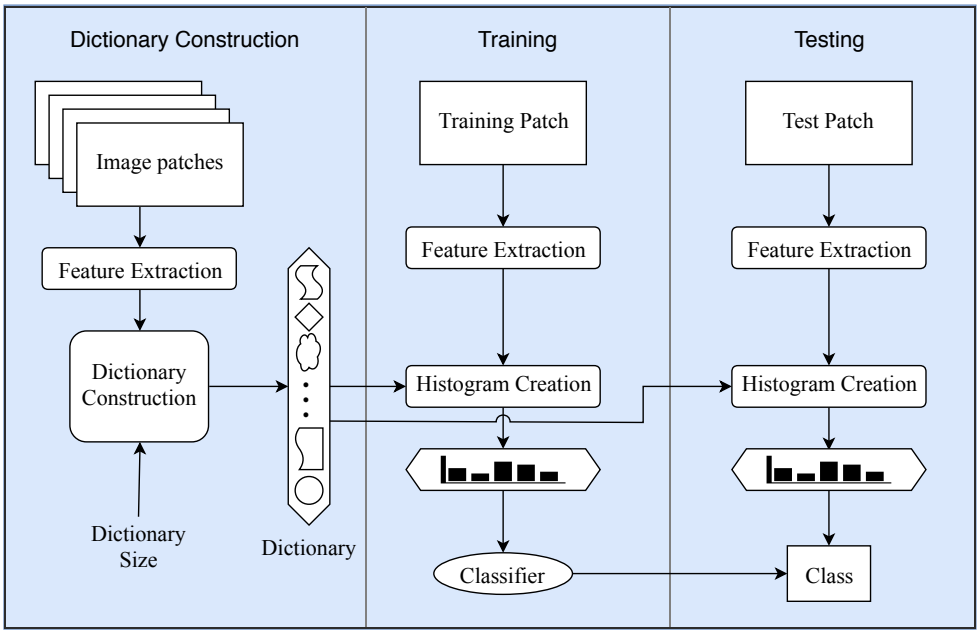



Figure 2.3: Bag of Visual Words technique

Although the first method has high accuracy, its time complexity is very high. On the other hand, Viola Jones framework is designed to work on real-time scenarios with the help of early rejection technique that will be explained in detail. The approach uses Haar-Like features since they are very easy to compute. This features are composed of additive and subtractive blocks that sums up to zero. With different types, scales and positions, these features can count to more than 100.000 which will dramatically slow down the system. Hence, Adaboost algoritm [32] is used to select most useful features. Finally, many weak classifiers are trained using them, and com-

bined to form a cascaded strong classifier. In testing phase, each stage of the cascaded classifier is responsible for either rejecting the object proposal or propagating it to the next stage. If a proposal passes through all stages it is accepted as a detection. Otherwise, it is rejected as early as possible. This eliminates the complex executions for background patches in the sliding window approach. Therefore, it results in a real time performance.

### 2.1.2 Single Stage Detectors

Single stage object detectors [61, 79, 80, 94] are the network architectures that reports the bounding boxes for objects directly from the image pixels. These methods are known to have high frame rates since the coordinate computation and classification is done directly in a single stage.

In the study of Sermanet et al. [94], a feature extractor network OverFeat and an object detection framework using it has been proposed. After extracting the features from the convolutional network, a fully connected network is applied to the feature maps in a sliding window manner to predict the coordinates of bounding boxes with a regressor and to classify the image patch. Since this approach results in multiple boxes around detected objects, the boxes related to an object are merged by averaging them.

In Single Shot Multi Box Detector (SSD) [61], multiscale feature maps are computed and bounding box coordinates are computed using default bounding boxes (aspect ratio, position, etc.) in different resolutions by estimating the offsets relative to the default boxes. Finally the class of the objects are estimated. The You Only Look Once (YOLO) approach [79, 80], is very similar to SSD in terms of prior box usage in the computations of extents. However, it uses a spatial grid to estimates the coordinates. This approach is explained throughly in Chapter 3.

### 2.1.3 Two Stage Detectors

Convolutional neural networks (CNN) are known to be very successful image classification problem. The idea in two stage detectors [21, 34, 35, 43, 47, 57, 83], is to use this power for classification as in the classical methods (see Section 2.1.1). However, the sliding window approach requires to classify extensive amount of image patches, and classification with CNNs is very expensive. Hence, the number of candidate windows must be decreased dramatically.

The first attempt with this paradigm, the study of Girshick et al., Regions with CNN features (R-CNN) [35], tries to eliminate negative windows before the classification stage. To this end, it uses a segmentaion method called selective search [104] to detect the regions that are most likely to have objects. The regions proposed by selective search are classified by a CNN after warping them to the input shape. Finally, coordinates and extents of the proposals are refined by bounding box regression.

Although R-CNN decreases the number of regions to test dramatically, it runs the CNN again and again for each proposal. However, SPP-net [43] computes feature maps from entire image only ones, and pools them from arbitrary image regions proposed by selective search. This operation is done by spatial pyramid pooling (SPP) which pools features with different grid sizes and combines them to create a fixed length representation. This allows regions with different sizes to be classified with a single classifier. Similarly, in Fast R-CNN [34], a region of interest (ROI) pooling layer is proposed to convert different sized feature maps to a fixed length vectors. In this approach, a region is basically divided into a 2D grid and pooling is applied to each cell. Unlike SPP, ROI pooling enables the Fast R-CNN to be trained end-to-end since it can backpropagate the error signal to the preceding layers.

Another important improvement comes with the Faster R-CNN [83] architecture. In this study, Ren et al. introduce a region proposal network (RPN) that eliminates the need for a separate region proposal stages. RPN is a fully convolutional network that is slid over the image. For each position, it makes a bounding regression utilizing prior knowledge about bounding boxes, called anchor boxes, and applies a binary

classification to decide on whether the predicted box contains an object or not. The remaining part of the architecture is the same as Fast R-CNN.

## 2.2 Object Tracking

The aim of object tracking is estimating the state of a moving target (or possibly multiple targets) through time in an image sequence. The state may include position, appearance, shape, velocity, etc. In simple terms, it is estimating the next state of an object given noisy observations up to the current time instance. In multiple objects case, estimations should be done for all targets.

### 2.2.1 Single Object Tracking

Before going into details of the related work on this topic, it is useful to make formal definition. As the name suggests, there is only one target to track in single object tracking problem. It can be formulated as a Bayesian network. In order to estimate the state $x_t$ utilizing all of the observations $z_{1:t}$ up to time instance $t$, one should find a solution that maximizes posterior probability $P(x_t|z_{1;t})$. Recursive Bayesian filter offers a solution with two stages. In prediction stage a motion model and the previous posterior estimate (as prior knowledge) are used to predict $P(x_t|z_{1;t-1})$. Then the actual posterior probability is computed with the help of an observation function and the prior prediction.

**Statistical methods:** With the assumptions that probability density functions are Gaussian and state transitions are linear, Kalman filter [52] offers a solution to Bayes equations. Extended Kalman filter [55] makes it possible to work with non-linear state transition models. However the Gaussian assumption has a negative effect on the success of filter. Particle filters [9, 26, 27] propose solutions that are based on Monte Carlo methods to work with non-Gaussian densities and nonlinear state transitions.

**Template matching:** Other than filtering based solutions to the equations, the posterior probability can be directly maximized. When the problem is to locate an object in the next frame, correlation filter based template matching [15, 66] and searching for the maxima of similarity [18, 19] can be applied. With the assumption that the target cannot move too much in consecutive frames, one can apply a search in the neighborhood to find the window with the best matching representation.

**Online learning:** Although statistical methods are robust to occlusions and noisy observations, they still need observations provided most probably by object detection or template matching. Object detectors and aforementioned template matching methods work with fixed appearance models. This has negative effects on the performance of statistical or template matching based tracking methods since illumination, scale and orientation changes, background clutter, and partial occlusions alter the appearance of the target significantly. This can be overcome by an adaptive appearance model [4, 10, 51, 86]. Ross et al. propose the Incremental Visual Tracking (IVT) method to keep track of the change in object appearance [86]. IVT uses an eigenbasis to represent a target, and particle filter to find the best matching window to the mean of the eigenbasis in the next frame. After accumulating a predefined number of windows, the eigenbasis is updated with incremental principle component analysis.

Another online tracking method is MILTrack algorithm [4] that utilizes multiple instance learning (MIL) [28] to learn the target online. At each time instance, the algorithm first crops patches around the target and classifies them with the MIL classifier to find the best matching one. Then, positive and negative bags formed with samples around the target are used to train weak classifiers. The most discriminative ones among the previous and new weak classifiers are chosen to form the strong classifier.

A breakthrough comes with the Minimum Output Sum of Squared Error (MOSSE) filter [10] which adds appearance model adapting ability to aforementioned correlation filter techniques. The earlier ones require strong constraints or complex training procedures. With a simple training procedure and unconstraint tracking method, MOSSE increases execution speed and accuracy. The appearance model is correlated with the next frame in frequency domain which makes it real time. The maximum

value in the correlation output is determined as the new location, and online update is applied. Although many studies follow the MOSSE approach, only a few of them are included here. Kernelized Correlation Filters [45] utilize nonlinear kernels having circulant structure which encode convolution of vectors. Correlating with such filters is similar to applying a classifier on different subwindows. Spatially Regularized Discriminative Correlation Filters (SRDCF) [24] apply regularization on correlation coefficients depending on their spatial locations to eliminate unwanted boundary effects caused by the assumption of periodicity in correlation filters. Discarding the background patches, and instead learning from shifted patches of the cropped target may cause suboptimal results. Background Aware Correlation Filters [33], however, exploit all background patches as negative examples for learning a filter which is more discriminative to background clutter.

Another important framework is Tracking-Learning-Detection (TLD) [51] that is capable of long term tracking thanks to its integration of tracking and detection. When a frame comes, it estimates the target location with both tracker and detector. While the former one uses temporal information, the latter utilizes spatial information. Finally, estimations are integrated to decide on actual result, and the detector is updated with positive and negative samples around the output. Learning phase makes the detector stronger while detector resets the tracker if it fails to avoid drifting.

**Neural network based tracking:** Similar to several computer vision problems, visual object tracking has benefited from the undeniable performance of deep neural networks. However, an extensive literature review has resulted in no new paradigms other than the classical methods. Similar to statistical methods, Ning et al., have proposed a method [74] that utilizes an LSTM to model temporal constraints (i.e., motion of the object).

Since CNNs have an excellent representation capability, online learning methods have been developed for them [73, 106, 107]. Teng et al., have proposed Temporal-Spatial Network [102] that resembles TLD approach. A feature network forms a representation of the target while a temporal network encodes the trajectory. With local spatial object information, spatial network refines the tracking state. Both temporal and

spatial networks are updated if necessary. Memory Augmented Visual Object Tracking [60] is another online learning approach that utilizes foreground and background memories. At each time step, foreground and background patches are written into memory. With patches in a region of interest, memory reads are done to create foreground and background heatmaps. Combining them gives the bounding box of the object.

Similar to other classical methods, there are studies that brings deep learning and template matching together. Siamese architectures [8, 41, 101] have been utilized in order to create matching functions between target and search image. Guo et al., add online learned target variation and background suppression transformations [41] to the siamese architecture. Similarly, correlation filters have been married with deep learning by designing filters working with deep CNN features, utilizing hierarchical information in CNNs, ensembling weak trackers from CNN layers or choosing between them with attentional mechanism [16, 23, 25, 65, 78].

### 2.2.2 Multiple Object Tracking

Tracking multiple objects is different than tracking a single one in terms of its requirements since it should retain object identities in addition to their trajectories. When there are multiple objects searching each one in the next frame is unreasonable since it requires the number of objects to be fixed. To this end, the widely used approach is tracking by detection instead of detection free methods whose single object counterparts has been discussed in Section 2.2.1. Hence, Luo et al. formulates the problem in a general perspective, following the tracking by detection approach in their extensive literature review [64] as maximal a posteriori estimation of all object's states through time which is given in Equation (2.1);

$$\hat{\mathbf{S}}_{1:t} = \arg\max_{\mathbf{S}_{1:t}} P(\mathbf{S}_{1:t}|\mathbf{O}_{1:t}), \tag{2.1}$$

where $\mathbf{S}_t = (\mathbf{s}_t^1, \mathbf{s}_t^2, ..., \mathbf{s}_t^{M_t})$ is the collection of object states ($\mathbf{s}_t^i$ of $i$-th object), $\mathbf{O}_t = (\mathbf{o}_t^1, \mathbf{o}_t^2, ..., \mathbf{o}_t^{M_t})$ is the collected observations ($\mathbf{o}_t^i$ for $i$-th object) and $M_t$ is the number of objects in $t$-th frame. This formulation converts the problem into data association

problem where the detections are matched to the states of objects being tracked. As reported in [64], this is done in two ways; as a probabilistic inference solution [11, 54, 71, 82, 85, 109, 111] and as a deterministic optimization solution [3, 7, 12, 13, 29, 48, 58, 70, 77, 96, 110, 115].

Similar to many other computer vision tasks, multiple object detection also exploits fabulous performance of deep learning methods. In order to learn a similarity metric, siamese networks [56] and quadruplet structure similar to siamese architecture [100] are utilized. However, these methods require an external trajectory creation algorithm and cannot utilize long term history of objects since they only focus on the similarity of objects and detections. Similarly, Sadeghian et al. [91] combine temporal features using multiple cues (appearance, interaction and motion) with RNNs to feed a target RNN that computes a similarity score which is used in bipartite graph matching. Milan et al. [69] use a single custom RNN for Bayesian state estimation, track birth/death and data association. However, their approach is not aware of the appearance of the objects. Fernando et al. [30] combine generative adversarial networks [38] with LSTM to detect objects, and use attention mechanism along with an LSTM to create short term trajectories which are associated with the detections if their positions intersect. Recurrent autoregressive network architecture proposed in [30] utilizes an external memory to hold previous inputs and internal memory for data association. Jiang et al. [50] formulates tracking as a reinforcement learning problem. After making predictions, trackers are associated with detections using an LSTM. An important drawback of the studies [30] and [50] is that they both require a tracking network for each object, which limits the number of objects and makes it necessary to have an external controller for the networks.

## 2.3 mUAV Detection and Tracking

Although the problem of detecting and tracking mUAVs is not a well studied subject, there are some attempts to mention. Mejias et al. [68] utilized morphological pre-processing and Hidden Markov Model filters to detect and track micro unmanned planes. Gökçe et al. [36] used cascaded boosted classifiers along with some lo-

cal feature descriptors. In addition to this pure spatial information based methods, spatio-temporal approaches exist. Rozantsev et al. propose a method that first creates spatio-temporal cubes using sliding window method at different scales, applies motion compensation to stabilize spatio-temporal cubes, and finally utilizes boosted tree and CNN based regressors for bounding box detection [87].

Drone-vs-bird challenge [17] has shown that when the conditions are harsh; e.g., camera is moving, drone is at a distant location, or there are distracting objects, classical methods fail to detect the drone while deep learning approaches have promising results. The successful methods [31, 92, 93] sent to the challenge other than the one explained in Chapter 3 utilize or get inspiration from Faster R-CNN [83].

A through literature survey has resulted in no study other than [112] for drone tracking. In that one, the authors have proposed a convolutional long short-term memory network that makes use of spatio-temporal information to simultaneously detect and track flying small objects.

## 2.4 Recurrent Neural Networks and Variants

Recurrent neural networks (RNN) [89] are the models that have a hidden state between the input and the output which is updated through time to hold the context of an input sequence. This enables them to learn a specific task instead of mapping input to output with feature learning as in the feed forward networks. It has been shown that RNNs are Turing-Complete [97], which means that they can mimic any program if proper weights can be found. They are mostly used to model sequences such as time series data, text, video and audio data. Architecture of a basic RNN can be seen in the left part of the Figure 2.4. The most important part in the architecture is the weight sharing mechanism. Unlike the feed forward networks, RNNs use same parameters at each stage of the computation. While training an RNN, Back Propagation Through Time (BPTT) algorithm [72, 84, 108] is utilized. In this method, the network is unfolded through time with the same parameters at each time step (see Figure 2.4). Although unfolding is done to a specific time step, there is no limitation on the

sequence length due to the recurrent structure of the hidden state. The training data for the network is a sequence of $< x_t, y_t >$ pairs. The hidden state should be initialized before each sequence which is done with all zeros in general. Then, outputs are computed according to Equation (2.2)

$$h_t = \sigma_h(W_x x_t + b_x + W_h h_{t-1} + b_h),$$
$$y_t = \sigma_y(W_y h_t + b_y), \tag{2.2}$$

where $\sigma_h$ and $\sigma_y$ are nonlinearities such as sigmoid or tanh functions. Loss is computed using the ground truth labels and the predicted outputs and back propagated through time to update the same parameters at each time step.

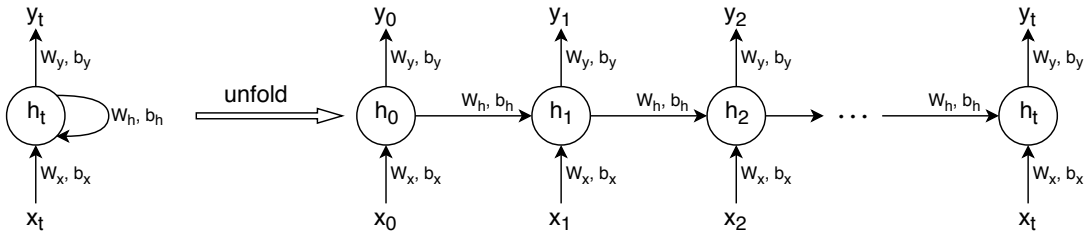



Figure 2.4: Basic RNN architecture, weight sharing and unfolding mechanisms. Adapted from: [37].

There are four different modes of RNNs; one to many, many to one, many to many and synchronized many to many. The left most diagram in the Figure 2.5 is a simple feed forward network without any recurrent structure. The second one gets only one input but outputs a sequence. This can be used to create text by providing the first word and expecting the remaining word sequence from the network. The third one is the other way around. It gets a sequence of inputs, and provides a single output. This can be used in problems like video and audio classification. The fourth diagram represents a mode that gets a sequence of inputs and provides a sequence of outputs. This mode can be utilized to solve machine translation problem. The last mode is the synchronized many to many which gets sequence of inputs and provides an output for each input at the same time step. This mode can be utilized to classify each frame of a video.

As mentioned before, RNNs can mimic any program in theory. But setting proper

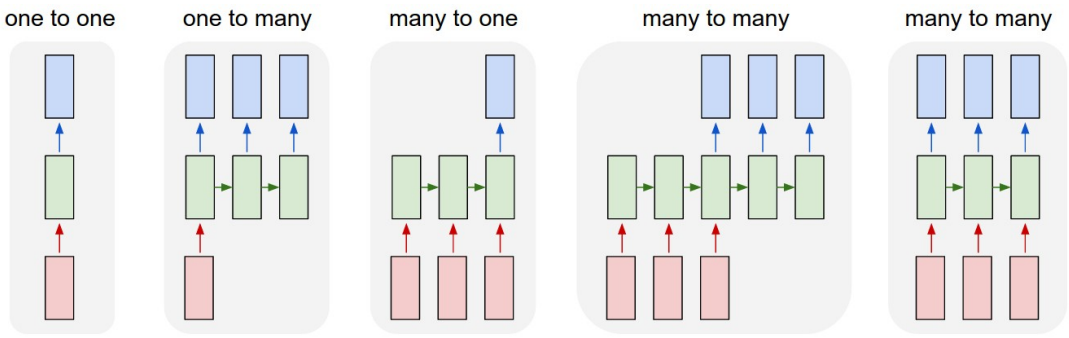



Figure 2.5: Different modes of RNNs. Source: [53].

weights is not that easy in practice. Although BPTT makes gradient descent algorithm possible to be used to learn the parameters, there are still problems that need solving. When the sequence length is kept short in training, it results in poor performance since it cannot get the long-term dependencies. When it is long enough to learn long-term dependencies, exploding or vanishing gradient problems are encountered since the network becomes very deep in time. This requires introduction of a memory structure as in biological systems. To this end Hochreiter and Schmidhuber has proposed Long Short Term Memory (LSTM) [46] architecture which is a variant of RNNs.

LSTM adds four different units to the basic RNN. In order to provide a memory-like structure, it includes a cell unit which remembers some information through time steps. Forget gate is utilized to decide on which parts of the cell states will be got rid of. Input gate is responsible for deciding on what to keep and what to add to the cell state. Finally, the output gate choses the parts of cell state that will be used while creating the output. With all of these units, basic RNNs gain the ability of remembering long term dependencies, getting rid of unnecessary details, selecting important parts from the input and remembered information for output. Internal structure of an LSTM cell can be seen in Figure 2.6.

For each time step $t$, there are three inputs to an LSTM cell which are $x_t, h_{t-1}$ and $C_{t-1}$. The outputs are $C_t$ and $h_t$. While computing the outputs, first stage is the decision of which parts will be forgotten from the cell state using the Equation (2.3)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{2.3}$$

where $\sigma$ is sigmoid function, $W_f$ and $b_f$ are network parameters. Then, the values

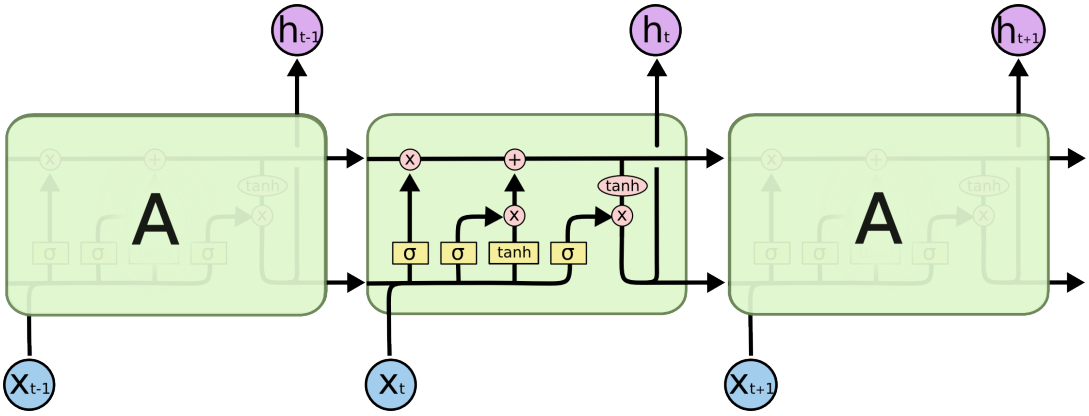



Figure 2.6: Chain of LSTM cells where yellow rectangles are neural network layers and red ellipses are pointwise operations. Joined arrows means vector concatenation, and arrow split means same copies of vectors. Top horizontal input and output arrow represents the cell state, whereas bottom one represents the hidden state. Adapted from [75].

that will be updated are determined with Equation (2.4), and the candidate content to be added to those values is computed using the Equation (2.5).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i). \tag{2.4}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c). \tag{2.5}$$

Having computed the forget and input gates, the new cell state can be created with (2.6)

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{2.6}$$

where the first term in summation eliminates the unnecessary content from the previous cell state, keeps the related information, and the second term adds new content from the input $x_t$. The final stage is the computation of new hidden state with the help of output gate as in Equation (2.7)

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t). \end{aligned} \tag{2.7}$$

To sum up, LSTM introduces mechanisms to carry, add and discard information through time, which enables it to remember long-term information with a memory like structure to deal with long-term dependencies.

## 2.5 Neural Turing Machine

Although RNNs are Turing Complete in theory as stated in Section 2.4, it is not easy to train them for executing specific tasks. In order to solve this problem, Graves et al., have proposed an architecture called Neural Turing Machine (NTM) that adds a large addressable memory increasing the ability of RNNs to simplify the training for algorithmic tasks [39]. The architecture has two fundamental components which are a controller network and a memory consisting of fixed length data rows called memory "locations". With this memory and controller structures it resembles the Turing Machine and Von Neumann Architecture. The overall architecture can be seen in Figure 2.7.

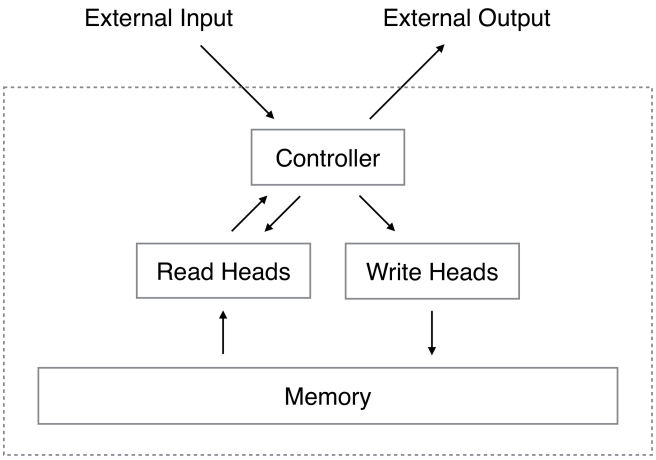



Figure 2.7: NTM architecture. Adapted from [39].

Controller of an NTM is responsible for the input from and output to the outside world, and interacting with the memory unit using read and write heads. During this interaction an attention based fuzzy addressing mechanism is utilized. Each head outputs a normalized weighting $w$ (see Equation (2.8)) over memory locations where each weight represents the contribution of each location to the read or write operation.

$$\sum_i w(i) = 1, \forall_i \, 0 \leqslant w(i) \leqslant 1. \tag{2.8}$$

The weighting can be seen as a fuzzy address to the memory locations which is based on content based addressing paradigm. While location based addressing focuses on

the memory locations and manipulates the data without knowing what it means, in content based addressing, related data itself is used as the address. Hence, the computation of weights outputted by heads uses the content address

$$w_t^c(i) \leftarrow \frac{\exp\left(\beta_t S[\mathbf{k}_t, \mathbf{M}_t(i)]\right)}{\sum_j \exp\left(\beta_t S[\mathbf{k}_t, \mathbf{M}_t(j)]\right)}, \tag{2.9}$$

where $w_t^c$ is the content based address, $\mathbf{M}_t$ is memory, and $\beta_t$ is the key strength that arranges the content focus precision at time $t$. Key vector $\mathbf{k}_t$ is the content created by the controller to access the memory, and the operator $S[\cdot, \cdot]$ is a similarity measure such as cosine similarity. The reader is directed to the original paper [39] for details of the addressing mechanism.

Having computed the weighting, read and write operations are easy to do. While reading, a weighted summation of memory locations is computed

$$r_t \leftarrow \sum_i w_t(i)\mathbf{M}_t(i), \tag{2.10}$$

where $\mathbf{M}_t$ is $N \times M$ memory matrix and $w_t$ is $N$ dimensional weight address vector. Writing is similar to content update in LSTMs. First, unrelated content is erased from the memory and new content is added to the remaining one since the fuzzy addressing mechanism holds different data in the same locations. This operation can be summarized as

$$\mathbf{M}_t(i) \leftarrow \mathbf{M}_{t-1}(i)[\mathbf{1} - w_t(i)\mathbf{e}_t] + w_t(i)\mathbf{a}_t, \tag{2.11}$$

where $\mathbf{1}$ is a vector full of 1's with the same dimension as the erase vector $\mathbf{e}_t$ and add vector $\mathbf{a}_t$. The multiplications with the memory locations are pointwise.

Putting it all together, NTMs make learning algorithmic tasks easier with the help of external long-term memory, reading and writing mechanisms.

# CHAPTER 3

# A SINGLE SHOT DETECTOR BASED MUAV DETECTION

The solution for drone detection problem described in this chapter is based on a single shot object detection model, YOLOv2 [80], which is the follow-up study of YOLO [79]. The model has been chosen because the motivation described in Chapter 1 requires real time performance and it is reported in [80] that the model outperforms others in both detection and time performance. The model has been adapted and fine-tuned to detect objects of two classes (i.e., drone and bird). Although the problem is detecting drones in the scene, bird class has also been included so that the network can learn robust features to distinguish them too. This adaptation has been used to participate in Drone-vs-Bird Detection Challenge [17], and ranked third.

The chapter continues with the details of network architecture and training details.

## 3.1 The Architecture

YOLOv2 is a fully convolutional network whose architecture tries to devise an end-to-end regression solution to the object detection problem. The architecture consists of two parts; convolutional feature extractor and object detector. The former has been designed with the knowledge gained from some of the most successful image classification networks in the literature. It has been shown with VGGNet [98] that stacking $3 \times 3$ convolution filters with max-pooling operations in between results in higher accuracies for image classification task and therefore representation learning. Hence, YOLOv2 makes use of such convolutional and max-pooling layers consec-

utively. "Network In Network" (NIN) [59] architecture proposes using multi layer perceptrons instead of linear filters in convolution operation. This results in a cross channel parametric pooling operation which can be implemented as a $1 \times 1$ convolution. It has been shown that, nonlinear convolution operations, which are done with multi layer perceptrons, can model the local patches better. Hence, the feature representations created by $3 \times 3$ convolution filters of feature extraction part of the network are compressed by $1 \times 1$ convolution filters. The first part is first pre-trained for classification task using global average pooling operation as suggested in [59], instead of flattening the final feature maps and using fully connected layers to predict the class. This prevents the classification layers from overfitting to data and works as a structural regularizer. While connecting the detector part to feature extractor, the classifier is detached, and higher and lower resolution features are concatenated as suggested in ResNet [44] to provide fine grained features. These features are required to correctly detect and localize smaller objects in the scene. The final architecture can be seen in Figure 3.1.

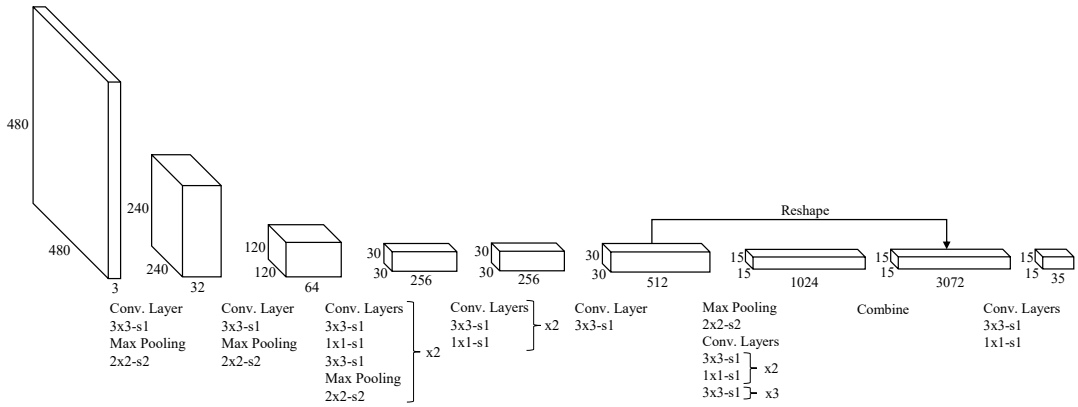



Figure 3.1: Adaptation of the YOLOv2 network for drone detection. Since the input size is changed, dimensions of succeeding layers are affected from that. All layers are fine-tuned with METU Drone Dataset (see Section 5.1).

In the detector part, final feature maps are passed through two convolutional layers to create an $S \times S$ grid where the duty of each grid cell is predicting bounding boxes of the form $(b_x, b_y, b_w, b_h, b_o)$. In this output, $b_x$ and $b_y$ are the coordinates of the centers of the boxes computed with respect to center of the grid cell, $b_w$ and $b_h$ are the

width and height in proportion to the whole image, and $b_o$ is the confidence that an object is in the bounding box. The final task of a grid cell is computing conditional class probabilities given the probability that the corresponding bounding boxes have objects in them. The mentioned confidence is defined as:

$$\text{Pr(Object)} * \text{IOU}(\text{truth, prediction}). \tag{3.1}$$

Prediction of positions with respect to grid cell instead of entire image makes it possible to bound ground truth values between 0 and 1. With the help of a sigmoid activation function, the problem turns into a constrained regression instead of unconstrained one. This makes it easier to learn and stabilize for the network. While predicting the width and height of bounding boxes, the model utilizes some prior information computed by K-means clustering on width and heights of ground truth bounding boxes. Redmon and Farhadi have reported that Euclidean distance is not a suitable metric for clustering bounding boxes since it over punishes larger boxes. In order to eliminate the dependence on size, they have proposed a new metric based on intersection over union (IOU) scores [80]. Likewise, the distance metric in Equation (3.2) has been used while creating prior information.

$$d(\text{box, centroid}) = 1 - \text{IOU}(\text{box, centroid}). \tag{3.2}$$

When it comes to how outputs are computed, it should first be mentioned that the network makes one prediction for each centroid computed with K-means clustering, at each cell of the output feature map. For the cell at offset $(c_x, c_y)$ from top left corner of the image and the centroid with prior width $p_w$ and prior height $p_h$, four relative coordinates $t_x, t_y, t_w, t_h$ and an objectness score $t_o$ are predicted. With all these data, the final outputs are predicted as:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ b_o &= \sigma(t_o). \end{aligned}$$

The final output size for a grid cell is:

$$\text{Output Size} = (N_{cls} + N_{coord} + 1) \times N_{cent}, \tag{3.3}$$

where $N_{cls}$ is the number of classes, $N_{coord}$ is the number of coordinates, $N_{cent}$ is the number of centroid bounding boxes used as prior knowledge and the 1 in the parenthesis is for the confidence value. In the adaptation, grid size is set to 15, number of classes is two, number of coordinates is four and number of centroids is five. Hence, the final output is of the shape $15 \times 15 \times 35$.

## 3.2 Training Details

The architecture has twenty-two convolutional layers in total, which has very high capacity, and is prone to overfitting. In order to achieve high accuracy with such deep models, one needs a large scale dataset that includes many scenarios of the problem, to get better generalization and avoid overfitting. However, the dataset provided by the challenge organizers was too scarce. To this end, an artificial dataset, named METU Drone Dataset, has been created including real drones, real birds and real backgrounds, which is described in detail in Section 5.1.

In order to get better initial points, commonly used pretraining and finetuning techniques has been applied as described in Section 3.1. In order to apply fine tuning, the network is initialized with the pre-trained weights using the ImageNet dataset [90] for image classification problem. Then the challenge dataset and the METU Drone Dataset have been divided into training (85%) and validation (15%) parts. The training part of the former one is duplicated four times before combining them to training sets since it is too scarce compared to the artificially created, large scale one. Then, the network is fine-tuned for 10,000 iterations with 128 as batch size and batch normalization after all convolutional layers.

The loss function used in iteration $t$ while finetuning the network for drone detection

is given in Equation (3.4):

$$
\begin{aligned}
loss_t = \sum_{i=0}^{W}\sum_{j=0}^{H}\sum_{k=0}^{P} \quad & \mathbb{1}_{noobj}\,\lambda_{noobj} * (-b_{ijk}^{o})^2 \\
& + \mathbb{1}_{early}\,\lambda_{prior} * \sum\nolimits_{r\in(x,y,w,h)} (prior_{ijk}^{r} - b_{ijk}^{r})^2 \\
& + \mathbb{1}_{k}^{truth}\,( \quad \lambda_{coord} * \sum\nolimits_{r\in(x,y,w,h)} (truth^{r} - b_{ijk}^{r})^2 \\
& \qquad + \lambda_{obj} * (\mathrm{IOU}_{truth}^{k} - b_{ijk}^{o})^2 \\
& \qquad + \lambda_{class} * (\textstyle\sum_{c=0}^{C} (truth^{c} - b_{ijk}^{c})^2)), \qquad (3.4)
\end{aligned}
$$

where $W$ is width of the grid, $H$ is the height, $P$ is the number of prior boxes, $\mathbb{1}_{noobj}$ means the box is not responsible for any object, $\mathbb{1}_{early}$ means the iteration is in the early stages of the training (i.e., $t < 12800$), and $\mathbb{1}_{k}^{truth}$ means bounding box $k$ is responsible for truth box. The coefficients have been set as follows; $\lambda_{noobj} = 1$, $\lambda_{prior} = 0.01$, $\lambda_{coord} = 1$, $\lambda_{obj} = 5$, and $\lambda_{class} = 1$. The first term tries to minimize the objectness score of a bounding box when it does not contain any object. The second aligns the predictions with the prior information for the early stages of training. The last contributes to the loss if the box k is responsible for an object indicated by a "truth" box. This term is divided into three parts which are responsible for coordinates, objectness score and classification error.

### 3.3 Detection Details

Although $480 \times 480 \times 3$ resolution was used as input size in training (see Figure 3.1), it is increased to $800 \times 800 \times 3$ in detection configuration. This is applicable since the network is fully convolutional. This increase is helpful in detecting small sized targets.

The aforementioned challenge requires the algorithm to work on a video sequence and report a bounding box for the only drone in the scene for each frame. Hence, the detection stage has been designed to report the bounding box which the network is most confident that there is the drone. Since the network is trained with two classes, the bird detections are eliminated after getting all predicted bounding boxes from the

last layer. Then a threshold, which can be determined according to accuracy on a validation set, is applied on the confidence values for objectness. If this operations eliminate all predicted bounding boxes, it means that the frame does not include a drone or it is not clear enough to detect. Otherwise, the one that has the highest confidence is selected as the prediction. Note that the algorithm can easily be extended to multi-drone situations with more intelligent thresholding strategies for different use than the challenge requires. One possible problem with this approach is encountered when the network mixes up a bird with the drone. If the objectness confidence of it is higher than that of the drone, it is selected as the prediction. In order to decrease the number of such misinterpretations, a *limited ignorance approach* (see Algorithm 1) that uses time domain information has been proposed. After determining the bounding box that the network is most confident, its intersection with the rectangle having same center, three times the width and height as the predicted bounding box in the previous frame is checked, assuming that the drone cannot move more than its height or width in a single frame. If the rectangles intersect, we can accept the newly predicted one. Otherwise, we ignore the current prediction and report the previous one if the limit has not been exceeded yet. After exceeding the limit, we reset it and cancel the technique for the same number of frames. During this period, we report the current predictions directly. Likewise, when there is no predicted bounding box in the previous frame, we directly report the prediction in current frame.

**Algorithm 1** Limited Ignorance Approach.

**Input:** Video frames $V = \{f_i\}$, maximum tolerance $T_{max}$, threshold $\theta$.

**Output:** Predicted bounding boxes $B = \{b_i\}$ (for each frame only the best box is reported).

$limit \leftarrow T_{max}$

$disabled \leftarrow FALSE$

**for** $f_t \in V$ **do**

    $D_t \leftarrow DETECTIONS(f_t)\ s.t.\ \forall_{\hat{b}_k \in D_t} \hat{b}_k^c = drone \wedge \hat{b}_k^o > \theta$

    $\hat{b}_{best} \leftarrow \hat{b}_j \in D_t\ s.t.\ \forall_{\hat{b}_k \in D_t} \hat{b}_j^o \geqslant \hat{b}_k^o$

    $\hat{b}_{ext} \leftarrow (b_{t-1}^x, b_{t-1}^y, 3 * b_{t-1}^w, 3 * b_{t-1}^h)$

    **if** $\hat{b}_{best} \cap \hat{b}_{ext} \neq \varnothing$ **then**

        $b_t \leftarrow \hat{b}_{best}$

    **else if** $disabled = FALSE$ **then**

        $b_t \leftarrow b_{t-1}$

        $limit \leftarrow limit - 1$

        **if** $limit = 0$ **then**

            $disabled \leftarrow TRUE$

            $limit \leftarrow T_{max}$

    **else**

        $b_t \leftarrow \hat{b}_{best}$

        $limit \leftarrow limit - 1$

        **if** $limit = 0$ **then**

            $disabled \leftarrow FALSE$

            $limit \leftarrow T_{max}$

# CHAPTER 4

# NEURAL TURING TRACKER: OBJECT TRACKING BASED ON NTM

In this chapter, an NTM based end-to-end object tracking model is proposed. The proposed architecture extends the abilities of NTM with a specialized memory layout, customized read/write heads and a novel adaptive Kalman filter based on multi layer perceptrons. The model is named Neural Turing Tracker (NTT).

## 4.1 Motivation

The insight gained from the drone detection method described in Chapter 3 has revealed the need for considering time domain to increase accuracy while working with videos. This means that it is required to work with tracking algorithms. Since real-time scenarios include several drones appearing in the scene at the same time, the method should be able to handle multiple objects.

The studies reviewed in Section 2.2 focus on only one part of the problem. Some try to refine the detections with statistical filtering while some others search for a similar appearance in consecutive frames. Another group of studies try to keep track of long term history of a single object. Single object tracking methods constitute such groups which do not make use of advantages of others. For multiple object case, most of the benefits of single object tracking methods have been discarded, and the focus is only on matching the detections with objects. This grouping of tracking methods unveils the requirements for a tracking method to be robust. These can be summarized as:

- In order to handle occlusions and failed detections, one needs a statistical mo-

tion model.

- In order to handle changes in the appearance of the objects, one needs online learning.
- In order to be able to do long term tracking, handle objects leaving and re-entering the scene, one needs a long term memory.
- In order to deal with multiple objects, one needs a matching mechanism and a controller that monitors tracking.

The requirements can be put together to form a general approach for multiple object tracking as given in Algorithm 2. As described earlier, an extensive literature review

**Algorithm 2** General Multiple Object Tracking Approach.

**Input:** Video frames $V = \{f_i\}$ for time instances $i = 0, ..., T$
**Output:** Predicted bounding boxes $B = \{b_i^j\}$ for each object $j$ in frame $i$

$L \leftarrow \varnothing$ (Empty set of trackers)
**for** $f_i \in V$ **do**
    $D_i \leftarrow DETECTIONS(f_i)$
    **for** Object $o \in D_i$ **do**
        **if** There is a tracker $t \in L$ that $o$ can be assigned as visible **then**
            Report $b_i^o$ using $t$ and $o$
            Update model and parameters of $t$ with $o$
        **else**
            Create a new tracker $t$ with $o$
            Report $b_i^o$ using $t$ and $o$
    **if** There are trackers $T \subseteq L$ that are not updated **then**
        Report bounding boxes with recently updated trackers $T_U \subseteq T$
        Update $T_U$
        Kill the remaining trackers $T_R \subseteq T$

has not resulted in a study that focuses on all of the requirements at the same time as

an end-to-end approach. This is because current architectures have been designed for specific requirements of the problem. However, NTM architecture described in Section 2.5 is capable of learning execution of arbitrary algorithms. Its external memory can be utilized for long term history of objects. The controller can be made responsible for matching detections with tracks. Additional modules can be designed to ease the learning of tracking task. This is the motivation behind designing a NTM based multiple object tracking algorithm that has a long-term memory, read/write mechanisms, internal controller and a special statistical filter. To the best of knowledge gained by the literature review, NTM is the only architecture that is capable of all requirements as an end-to-end trainable network.

It is recommended to use curriculum learning strategy while training networks that are designed to solve algorithmic tasks [40, 81, 113, 114]. Curriculum learning [6] suggests starting training procedure with a simpler version of the problem and increasing the difficulty as training progress. Since the simplest form of the problem is single object case, a novel network architecture to solve single object tracking task is proposed in this chapter. In addition, a method for adapting the proposed architecture to multi object case is provided in Section 4.4. With this adaptation curriculum learning can be applied.

## 4.2 Overview

The proposed architecture has a recurrent structure since it processes image sequences. The NTM based architecture can be seen in Figure 4.1. A custom NTM cell is executed for each video frame. First, the controller is updated with object detection results. It is followed by a read operation from memory. If the read result does not contain a tracking information a new track is created following the green path. After refining the tracking information with Kalman filter, memory is updated following the blue path. With the same information, bounding boxes are computed as output.

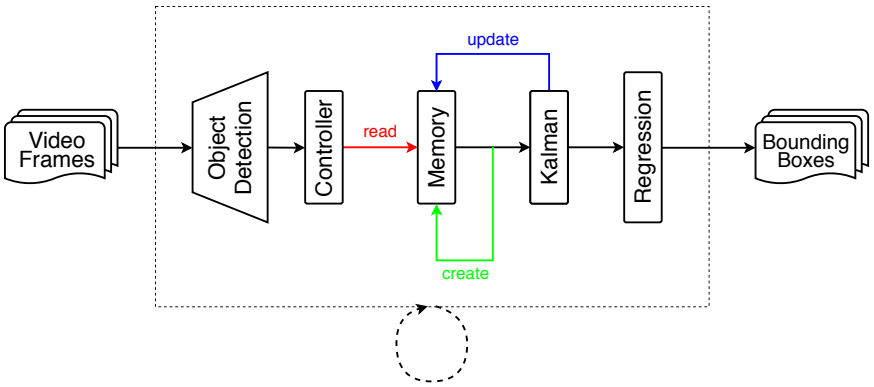



Figure 4.1: Overall architecture of the model.

## 4.3 Model in Detail

The proposed architecture has its roots from the Dynamic Neural Turing Machines (D-NTM) [40]. Starting from the original D-NTM and getting inspiration from its equations the final architecture has been designed as shown in Figure 4.2.

In this model, the controller is responsible for managing the input which is the concatenation of the detection bounding box and feature vector describing it. The controller prepares the input for content addressable memory. It utilizes its hidden state when detection fails but object still needs to be tracked. The read head is in charge of computing the address weights for the provided input using content addressable memory while the memory unit stands for long term storage of information. Kalman module is a novel adaptive Kalman filter that uses multi layer perceptrons instead of covariance matrices to decide on error of prediction and observation. Finally, the track creator and updater are write heads that are responsible for creating a new track when the current object is not tracked and updating the state of a track if the current detection has a corresponding tracker. Since the memory is content addressable, there is no need to an external matching algorithm. Memory read directly gives the tracking information related to the detection. The final module is a bounding box regressor that converts the internal state into the external state which is the rectangle that bounds the object.

Controller is a simple LSTM that gets input $i_t$ at time $t$ and previous hidden state $h_{t-1}$

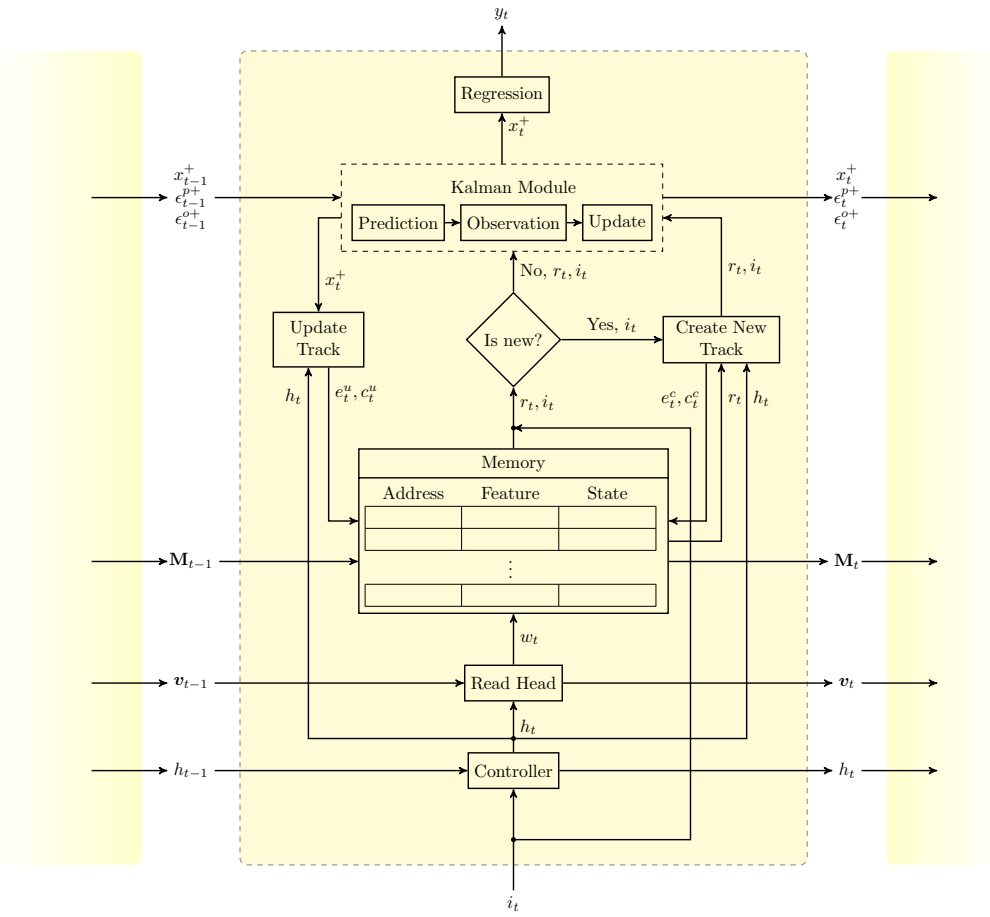



Figure 4.2: Proposed novel single object tracking architecture that is based on D-NTM.

to update its internal state. Similarly, bounding box regression module is a simple multi layer perceptron. In order to implement decision box that divides the data flow in two different paths, two different methods can be proposed. The first one which is rather easier is using an on-the-fly computation graph creation framework such as PyTorch [76]. The other can be utilizing a gating mechanism that passes very little portion of data through blocked path and most of it through the desired path. The remaining part of this section explains the specialized and novel modules and how data flows inside network.

### 4.3.1 Read/Write Heads

The architecture represented in Figure 4.2 includes three heads, namely, read head, create new track and update track. The first one is responsible for computing the address weight vector corresponding to the tracking information in memory that is associated with the input. The others are write heads that are responsible for creating erase and candidate memory vectors that are used in memory write operations for creating a new track or updating an existing one. To summarize, the read head finds the corresponding memory location and the other parts of the architecture uses that weight vector for the remaining part of operations.

The addressing mechanism is the continuous one adapted from [40]. First, read head computes a key vector $\boldsymbol{k}_t$ for content based addressing,

$$\boldsymbol{k}_t = (\mathbf{W}_k)^\top \boldsymbol{h}_t + \mathbf{b}_k, \tag{4.1}$$

where $\boldsymbol{h}_t$ is the hidden state of controller, $\mathbf{W}_k$ is the weight matrix, and $\mathbf{b}_k$ is the bias vector. Then, sharpening factor $\beta_t \in \mathbb{R} \geqslant 1$ is computed with,

$$\beta_t = softplus((\mathbf{u}_\beta)^\top \boldsymbol{h}_t + \mathrm{b}_\beta) + 1, \qquad softplus(x) = log(\exp(x) + 1), \tag{4.2}$$

where $\mathbf{u}_\beta$ and $\mathrm{b}_\beta$ are weight and bias vectors trained to compute sharpening factor. Given $\boldsymbol{k}_t$ and $\beta_t$, logits for address weights are computed by,

$$\boldsymbol{z}_t[i] = \beta_t S(\boldsymbol{k}_t, \mathbf{M}_t[i]), \tag{4.3}$$

where $\mathbf{M}_t$ is the current memory content, and $S(\cdot, \cdot)$ is basically cosine distance. At this point, the logits can directly be used to compute address weights. However, utilizing a dynamic least recently used addressing scheme might be useful when an object that is not being tracked appears in the scene. Hence, one can follow the approach in [40]. First the exponentially moving averages of the logits ($\boldsymbol{z}_t$) are computed using its previous value and current logits,

$$\boldsymbol{v}_t = 0.1\boldsymbol{v}_{t-1} + 0.9\boldsymbol{z}_t. \tag{4.4}$$

In order to make it dynamic, a scaling factor $\gamma_t$ is used to adjust the influence of previously written memory locations on the weights of current time step,

$$\gamma_t = \mathrm{sigmoid}((\mathbf{u}_\gamma)^\top \boldsymbol{h}_t + \mathrm{b}_\gamma), \tag{4.5}$$

where $\mathbf{u}_\gamma$ and $\mathrm{b}_\gamma$ are the parameters of the function computing the scaling factor. Finally, weights are computed using updated logits,

$$\boldsymbol{w}_t = \text{softmax}(\boldsymbol{z}_t - \gamma_t \boldsymbol{v}_{t-1}). \tag{4.6}$$

As it is easily seen from Equation (4.6), when the scaling factor is zero, least recently used addressing is discarded. Otherwise, their usage is limited according to that scalar.

On the other hand, the write heads do not compute any address weights. They are responsible for updating the memory content by erasing unnecessary parts and adding the relevant information. Since both use the same mechanism, a combined representation can be used for the data they use. While creating a new track, one needs to make use of input $i_t$. On the contrary, update of existing track requires using posterior state estimate $\boldsymbol{x}_t^+$. Both are represented as $d_t$. The first thing that a write head should do is computing an erase vector that is discarded from the memory. A simple multi layer perceptron $\phi_e$ can model that function as,

$$\boldsymbol{e}_t = \phi_e(\boldsymbol{h}_t). \tag{4.7}$$

Then, a scalar gate $\alpha_t$ is computed based on controller's hidden state and the provided data $d_t$,

$$\alpha_t = \phi_\alpha(\boldsymbol{h}_t, d_t), \tag{4.8}$$

where $\phi_\alpha$ is a simple neural network. Finally, candidate memory content vector $\boldsymbol{c}_t$ is computed with,

$$\boldsymbol{c}_t = \text{ReLU}(\mathbf{W}_h \boldsymbol{h}_t + \alpha_t \mathbf{W}_d d_t), \tag{4.9}$$

where $\mathbf{W}_h$ and $\mathbf{W}_d$ are trainable weight matrices.

### 4.3.2 Custom Memory Layout and Operations

The proposed tracking architecture has an external memory $\mathbf{M}_t$ whose each cell consists of three parts, namely address, feature and state. The address part is a trainable vector that enables the network with combining location based addressing and content

based addressing. On the other hand, since the input of the network is the concatenation of a bounding box with features representing it, best way to hold the trackers in memory is keeping the features and internal state. They are together called the content part of the memory cell. Although they can be used as is, the equations can be updated to process them separately.

When it comes to operations, they are basically memory read and write. Read operation is just a single matrix-vector multiplication that corresponds to a weighted summation of memory cells. Equation (4.10) formulates the operation.

$$\boldsymbol{r}_t = (\mathbf{M}_t)^\top \boldsymbol{w}_t^r. \tag{4.10}$$

Memory write operation, however, is not that simple. It requires first erasing the irrelevant data and then adding the related one. Since the data is stored in the content part ($\mathbf{C}_t$) of the memory, only it is updated as seen in Equation (4.11),

$$\mathbf{C}_t[j] = (1 - \boldsymbol{e}_t \boldsymbol{w}_t^w[j]) \odot \mathbf{C}_{t-1}[j] + \boldsymbol{w}_t^w[j] \boldsymbol{c}_t, \tag{4.11}$$

where $\odot$ represents element-wise multiplication.

### 4.3.3 Kalman Module

The classical Kalman filters [2, 52, 55] consists of three steps to solve Bayesian estimation equations defined in Section 2.2.1. In the first one a prior state estimate is predicted with a motion model that uses previous posterior state estimate. Then, observation is done and its innovation is computed. Kalman gain is the measure of how reliable the prediction and observation. Since they use a Gaussian distribution assumption, they require working with covariance matrices that are responsible for representing the error in observation or prediction. Finally, the prior state estimation is updated with the help of Kalman gain to report posterior one. Akhlaghi et al. [2] propose a method to estimate process and measurement noise covariance matrices adaptively. Getting inspiration from their study, a novel filter is designed to solve Bayesian estimation problem in the same way with Kalman filter but with no assumptions about distributions. Hence, covariance matrix based computations have been replaced with neural network based trainable functions.

Similar to classical methods, the first step is prediction. As mentioned before, prediction is a function of previous posterior state estimation. In our scenario, previous state information is provided by two sources; directly from the previous posterior state estimation and current memory read that holds tracking information. Then, Equation (4.12) gives the prior state estimation for current time instance,

$$\boldsymbol{x}_t^- = \phi_1(\boldsymbol{x}_{t-1}^+, \boldsymbol{r}_t), \tag{4.12}$$

where $\boldsymbol{x}_{t-1}^+$ is the previous posterior state estimation and $\boldsymbol{r}_t$ is the current memory read vector. In a similar way, one can estimate the prior prediction error as in Equation (4.13),

$$\boldsymbol{\epsilon}_t^{p-} = \phi_2(\boldsymbol{\epsilon}_{t-1}^{p+}, \boldsymbol{r}_t), \tag{4.13}$$

where $\boldsymbol{\epsilon}_{t-1}^{p+}$ is the previous posterior estimate of prediction error.

The second stage is observation where the input bounding box is utilized to make a measurement, given in Equation (4.14), about the output bounding box.

$$\boldsymbol{m}_t = \phi_3(\boldsymbol{i}_t). \tag{4.14}$$

In this stage of computation, the prior observation error should also be predicted as in Equation (4.15),

$$\boldsymbol{\epsilon}_t^{o-} = \phi_4(\boldsymbol{\epsilon}_{t-1}^{o+}, \boldsymbol{i}_t), \tag{4.15}$$

where $\boldsymbol{\epsilon}_{t-1}^{o+}$ is the previous posterior estimate of observation error. The functions $\phi_i$ that are used in the first two stages are simple feed-forward neural networks.

The final stage is the update of prior state estimation using a gating mechanism to create the posterior state estimation and update of error predictions. To start with, the gate $k$ is computed as the fraction of prediction error in the total error,

$$\boldsymbol{g} = \boldsymbol{\epsilon}_t^{p-} \oslash (\boldsymbol{\epsilon}_t^{p-} + \boldsymbol{\epsilon}_t^{o-}), \tag{4.16}$$

where $\oslash$ is element-wise division. Then, the posterior state estimate is the weighted combination of prior state estimate and observation as given in Equation (4.17).

$$\boldsymbol{x}_t^+ = (1 - \boldsymbol{g})\boldsymbol{x}_t^- + \boldsymbol{g}\boldsymbol{m}_t. \tag{4.17}$$

Having predicted the posterior state estimate, one can update the prediction and observation errors as follows,

$$\begin{aligned} \boldsymbol{\epsilon}_t^{p+} &= |\boldsymbol{x}_t^+ - \boldsymbol{x}_t^-|, \\ \boldsymbol{\epsilon}_t^{o+} &= |\boldsymbol{x}_t^+ - \boldsymbol{m}_t|. \end{aligned} \tag{4.18}$$

With the final Equations (4.18), the error predictions are corrected, and the filter becomes ready for the next prediction.

### 4.3.4 Data Flow and Bounding Box Computation

The aim of the section is twofold; clarifying the way data is processed in the network, and demonstrating the correspondences between the proposed architecture and Algorithm 2.

First of all, Figure 4.2 represents the operations of the network in one time instance where the horizontal connections represents the signals that are coming from previous times instance and passed to the next one. However, the data corresponding to current frame flows from bottom to top. First, the controller gets the input and previous hidden state to update its internal memory which is passed to the next time step and the read/write heads. Then, read head computes the corresponding address weights that are used by all memory operations. Next, a memory read is executed to get tracking information. Comparing it with the input, one can decide on whether the object is being tracked or a new one. At this point, data flow is divided into two paths. If the object is new, a new tracker is created and its corresponding data is read from the memory. Otherwise, the corresponding tracking information has already been read from the long term memory. After passing the read vector and input to the Kalman module, two concurrent operations; updating the track and outputting the bounding box, finalize the operation for the current time instance.

When it comes to correspondences, one can examine the algorithm step by step. The external memory of the network starts execution as a zero matrix which corresponds to 4th line in the algorithm. The if-then-else statement in lines 8 to 13 matches with

the content based addressing, Kalman module, write heads and bounding box regressor. In a single object scenario, the last if statement is executed when the object detection fails. Hence, one can utilize a no-object token in such a case. Putting it together, examining the algorithm and proposed architecture simultaneously results in an observation that the network is capable of tracking objects if it can be trained.

## 4.4 Adaptation for Muliple Object Tracking

Although the motivation is for multiple objects, the proposed architecture is designed to work with a single object as explained in Section 4.1. However, it is very easy to adapt the model to multiple object case. NTT is basically an RNN with some extension adding it some important skills for learning tracking task. The aforementioned architecture works in the mode at the right most diagram of Figure 2.4. In this mode, it gets an input for each frame and provides a bounding box for it. Instead, one can utilize the other many to many (i.e., delayed many to many) mode such that all detections are written to the memory to update or create trackers, and bounding boxes are created afterwards using all trackers for a single frame.

In clearer words, when a frame comes bounding boxes are detected and given as input to the system sequentially. After the last box a special token is provided to denote that there is no other box to process. While each box is being processed, the network finds the corresponding tracker and updates it, or creates a new one. After the token is given, network starts to output all of the tracking information as predicted bounding boxes. Unmatched trackers may create output until a threshold is passed. Then, they are killed. Finally, the network outputs an end-of-boxes token which means that there will not be any other output for that frame. The following frames repeat the same steps.

# CHAPTER 5

# EXPERIMENTS AND RESULTS

In this chapter, an artificial dataset is described along with the novel creation method. The dataset is named METU Drone Dataset and published in [1]. In addition, the dataset provided by the Drone-vs-bird challenge [17] and evaluation metrics used while assessing the performance are described. With the help of them, the assessment of object detection method is done. In addition, the learning capability of the method is discussed. Finally, the comparison with other methods that were able to send meaningful results to the Drone-vs-bird challenge is done.

## 5.1 METU Drone Dataset

Drone flights have limitations due to inadequate battery technology, weather conditions and legislative regulations; hence, there is no publicly available large scale dataset for training deep networks for drone detection. However, the approach in Chapter 3 requires immense amount of data to learn useful features. One possible solution to this is creating an artificial dataset. To this end, public domain pictures of drones and birds, and videos of coastal areas have been collected. First, the backgrounds of the drone and bird images are deleted manually to create transparent backgrounds which makes it more realistic when overlaid on a different background scene.

While creating artificial images, an $R \times C$ grid is utilized. For each grid cell, images are created by placing drones in a random position in the cell with different sizes in predefined size intervals and different background scenes extracted from all videos.

This is done for each drone twice; once with only the drone and once with a randomly positioned bird. Figure 5.1 represents the creation of an artificial data example.

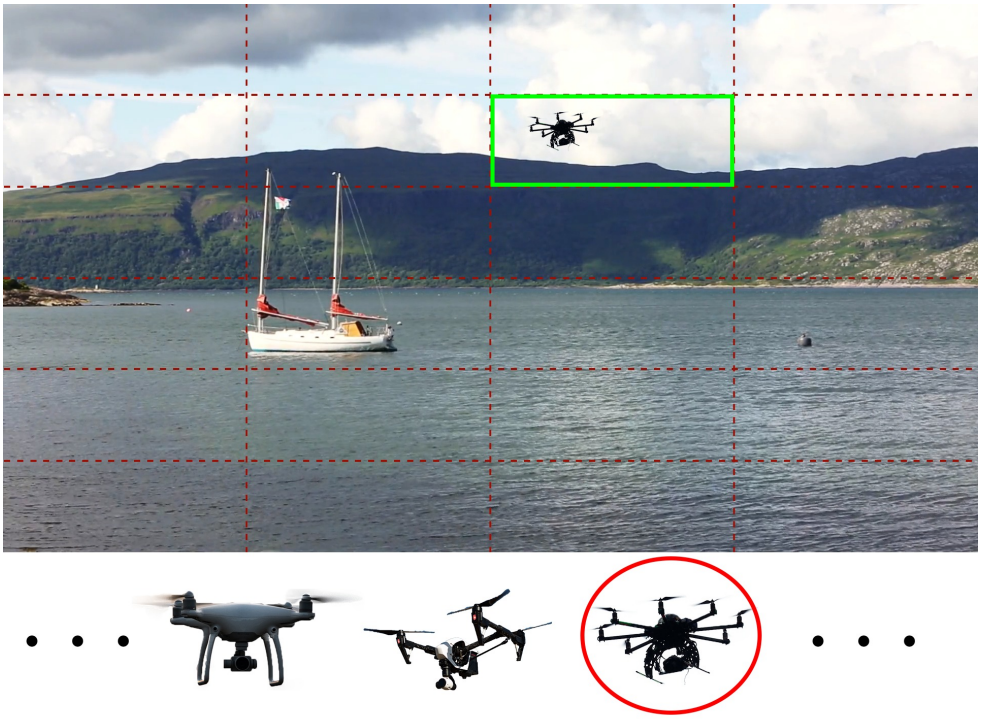


Figure 5.1: Creating a single artificial data example with a randomly chosen background scene and a drone with transparent background.

The overall dataset creation process is summarized in Algorithm 3. As can easily be seen, the dataset needs a huge storage size when all of the configurations are used. Hence, some portion of the configurations are eliminated with probability

$$p = 1 - \frac{Max.\ allowed\ size}{Total\ size\ for\ all\ configurations},$$

to reduce the size of the dataset to reasonable amounts.

The details of the created dataset can be found in the Table 5.1.

Fully randomized selections and configurations result in very different data examples. Most of the created images have realistic looking. This is an expected consequence because the background scenes are extracted from real videos, drones and birds are extracted from real photos taken while they are flying. Examples for realistic images can be seen in Figure 5.2. However, the randomization results in some problems.

**Algorithm 3** The algorithm applied for preparing the METU Drone dataset.

$S \leftarrow \{s_i\}$ predefined size intervals
$D \leftarrow \{d_i\}$ drone images (background eliminated)
$B \leftarrow \{b_i\}$ bird images (background eliminated)
$V \leftarrow \{v_i\}$ background videos
$R \leftarrow$ # of rows that the image will be divided into
$C \leftarrow$ # of columns that the image will be divided into
$G \leftarrow \{g_{i,j} \mid g_{i,j} = (r_i, c_j)\}$ $R \times C$ grid
**for all** $(d, g, s, v) \in D \times G \times S \times V$ **do**
    ignore this configuration with probability $p = 1 - \frac{Max.\ allowed\ size}{Total\ size\ for\ all\ configurations}$
    draw random positions $p_0$, $p_1$ and $p_2$ in $g$
    draw random sizes $s_0$, $s_1$ and $s_2$ from $s$
    draw random frames $f_0$, $f_1$, $f_2$ from $v$
    draw random birds $b_0$ and $b_1$ from B
    draw random positions $p_{b,0}$ and $p_{b,1}$ for birds in the scene
    draw random sizes $s_{b,0}$ and $s_{b,1}$ for birds where $s_{b,0}$ is from smaller half of $S$, and $s_{b,1}$ is from greater half of $S$
    overlay $f_0$ with $d$ resized to $s_0$ in position $p_0$
    overlay $f_1$ with $d$ resized to $s_1$ in position $p_1$
    overlay $f_1$ with $b_0$ resized to $s_{b,0}$ in position $p_{b,0}$
    overlay $f_2$ with $d$ resized to $s_2$ in position $p_2$
    overlay $f_2$ with $b_1$ resized to $s_{b,1}$ in position $p_{b,1}$
    save $f_0, f_1, f_2$ into the dataset

Since the algorithm does not consider semantic information, it may result in some unrealistic images. When the illumination source in the scene and illumination on the drone or bird has different characteristics, it results in an inconsistency. Another problem is that algorithm places the drones and birds at random sizes without considering the objects in the scene. For example, it seems very unrealistic when a drone near a human is two times bigger than the human. Figure 5.3 shows some unrealistic examples from the created dataset.

Table 5.1: Details of the dataset.

| Aspect | Information |
|---|---|
| # drones | 89 |
| # birds | 126 |
| # background videos | 11 |
| # rows in grid | 12 |
| # columns in grid | 10 |
| # size intervals | 19 |
| size intervals | in [5,160] (bias towards smaller values) |
| image resolution | $850 \times 480$ |
| # resulting images | 676,534 |

Figure 5.2: Realistic examples from the created dataset.

## 5.2 Drone-vs-bird Challenge Dataset

For the challenge, 5 MPEG4-coded videos were provided to participants. although the videos have only one drone that may enter and leave the scene at arbitrary time instances, there are several distractor objects (i.e., birds) in some of them. As can be seen from the examples in Figure 5.4, it is very difficult to deal with those diverse background and illumination conditions, clutter, different scales, viewpoints

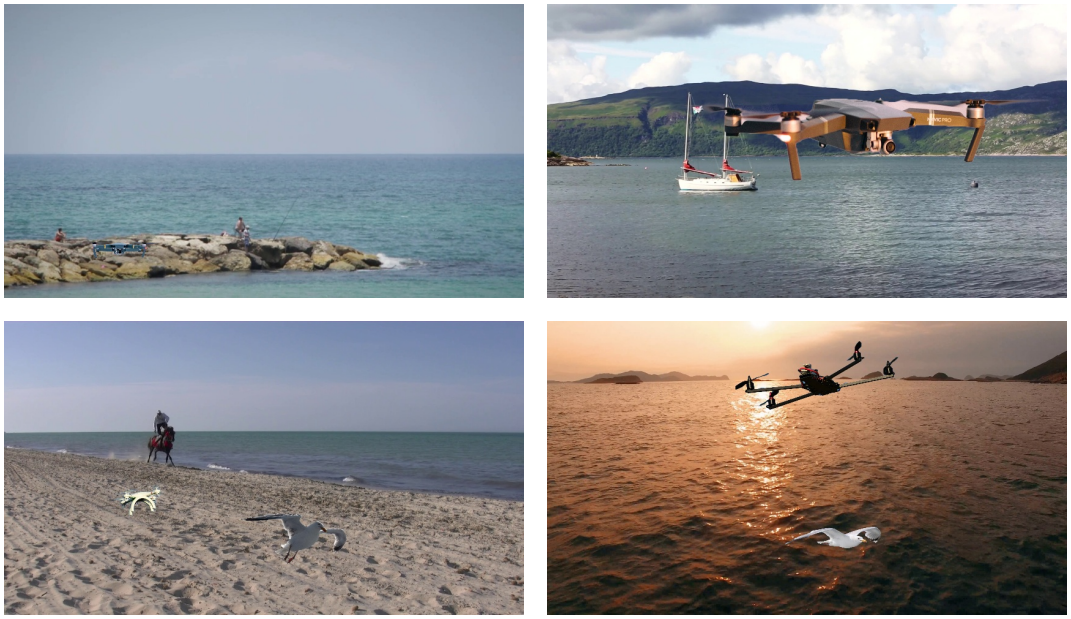


Figure 5.3: Unrealistic examples from the created dataset.

and inadequate contrast. It is apparent that this dataset cannot fulfill the expectations described in Section 5.1. Hence the detection algorithm does not fully depend on it. However, the only test video which is similar to the training ones provided by the challenge organizers has been used to assess the performance of the method.

## 5.3 Evaluation Metrics

In order to evaluate the detection method explained in Chapter 3, well-known precision-recall (PR) curve and the penalty metric defined in [17]. Precision is defined as

$$Precision = \frac{t_p}{t_p + f_p}, \tag{5.1}$$

where $t_p$ is is the number of true positives and $f_p$ is the number of false positives. This corresponds to the answer of the question that what fraction of retrieved elements; i.e., detected objects, is actually the wanted ones. In a related vein, recall is defined as

$$Recall = \frac{t_p}{t_p + f_n}, \tag{5.2}$$

where $f_n$ is the number of false negatives. It is a metric that measures what fraction of the relevant examples; i.e., ground truth boxes, is retrieved. When there is a matching


Figure 5.4: Examples from the Drone-vs-bird Challenge Dataset.

detection with a ground truth box, it is counted as true positive. If a detection does not match any of the ground truth boxes, it is a false positive detection. On the other hand, if there is no detection that matches a ground truth box, it means the box is classified as negative while it is actually positive. This is counted as a false negative. In order to decide whether a detection matches a ground truth box or not, the Intersection over Union (IOU) metric is utilized,

$$IOU(d, g) = \frac{A(d \cap g)}{A(d \cup g)}, \tag{5.3}$$

where $A$ is the area function, $d$ and $g$ are the detection box and ground truth box. In clearer words, it is the ratio of overlapping area of boxes to the area of their union. When they do not intersect, it becomes zero whereas it is one in case of perfect overlap. Therefore, the value $0.5$ is a good measure of overlap to decide on whether they match or not.

Another metric utilized in evaluation is the prediction penalty that measures the area of smallest rectangle that includes both the ground truth and predicted bounding boxes

divided by the area of ground truth bounding box. This division provides normalization that results in 1 as minimum penalty score when the perfect overlap occurs. The less accurate overlap, the more penalty score. This is because as the area of overlap decreases, the boxes diverge increasing the area of smallest rectangle that bounds both. This is exemplified in Figure 5.5.

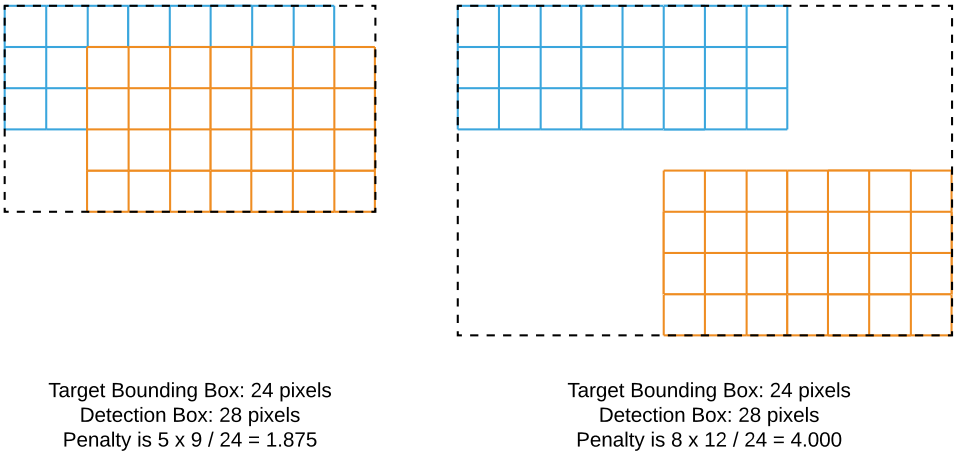



Figure 5.5: Examples for prediction penalty metric where blue rectangles represents the pixels of the ground truth bounding box and the orange ones are for the detection. Adapted from: [17].

The defined penalty metric is calculated for only one detection and ground truth box pair. In case of evaluating detection in videos, square root of the mean squared penalties of individual frames is utilized.

## 5.4 Results

As explained in Chapter 3, the detection network outputs a confidence value as a probability of including an object for each box, and a threshold is applied to eliminate detections that the network is not confident enough. In order to test the effects of the threshold value on detection performance precision and recall have been computed with different threshold values in range [0,1] since it is applied on a probability. The PR curve given in 5.6 shows that only a few configurations have resulted in poor performance. However, almost all of the configurations have achieved around 90%

precision and recall at the same time. In more clear words, the closer the PR curve to the top right corner, the better the performance of the method, and the evaluated method has a very good detection performance.

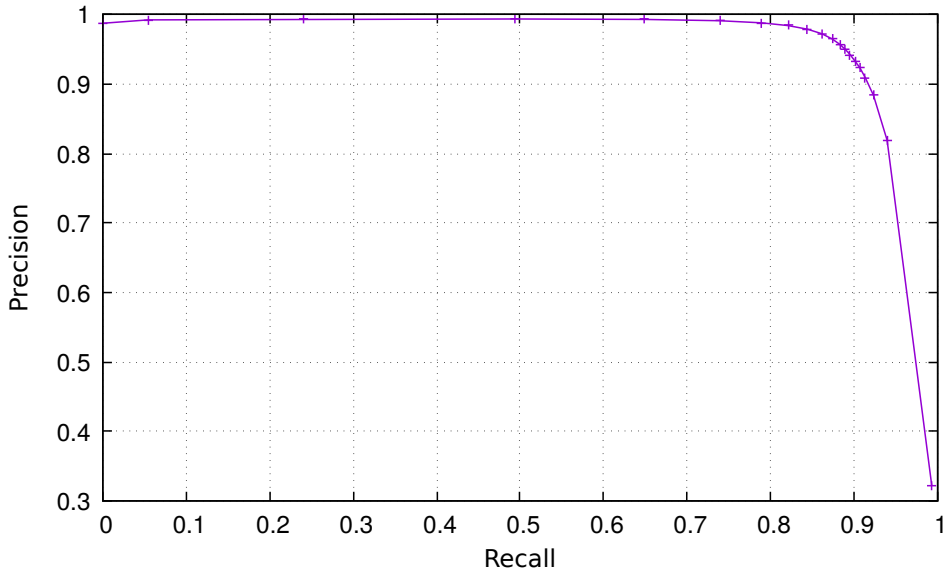



Figure 5.6: Precision-Recall Curve created with different confidence thresholds.

Another metric that has been utilized to test the effects of threshold value is prediction penalty. Figure 5.7 shows the change of average prediction penalty with respect to confidence threshold. The reason for higher penalties is that when the threshold increases detection rate decreases. When a drone cannot be found, the top-left pixel is reported as prediction, which is a requirement of the challenge. This results in a huge penalty since the metric is designed that way. Hence, the smallest possible threshold (which is zero) has been chosen for quantitative evaluation on the test video of the challenge. Although this threshold hurts precision in the artificial dataset, it works well in the provided test video. Analysis of the training procedure makes it possible to assess the learning capability of the network. As it can easily be seen from Figure 5.8, training loss sharply decreases from a sky-high point to reasonable amounts. Then, it decreases to almost zero which is the optimum point. The sharp decline implies that the network along with the created dataset has a strong learning capability.

As a final result to discuss, one can consider the penalty score of the successful meth-

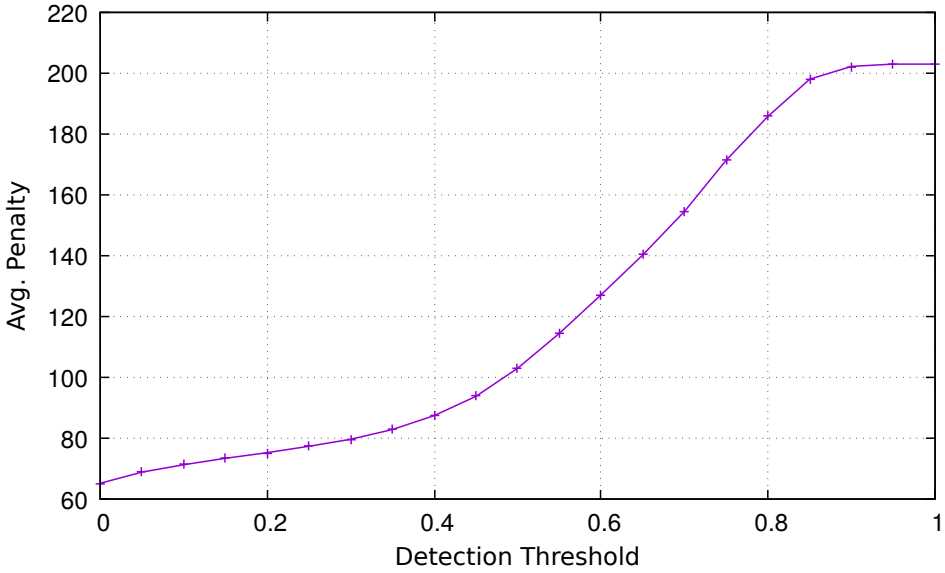



Figure 5.7: Change of average prediction penalty with respect to confidence threshold.

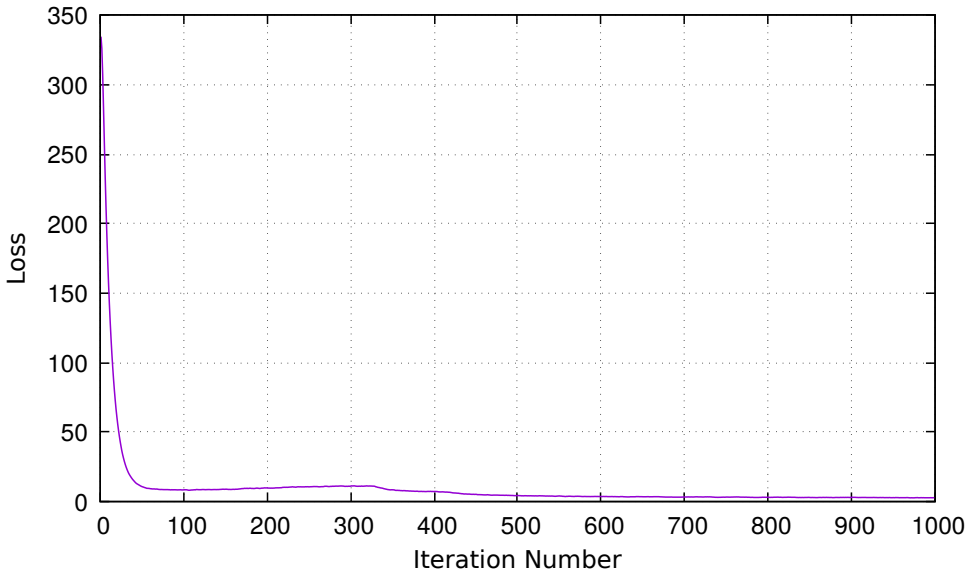



Figure 5.8: Change of loss in first 1000 iterations.

ods on challenge test video. Since the challenge has permitted the participants to send results from at most three settings of their algorithms three of the four successful participants have come up with three settings as seen in Table 5.2. The bold entries in the table corresponds to the method proposed in Chapter 3 where settings have different limits for ignorance algorithm which are respectively 2, 4 and 6.

Other challenge participants used similar methods to ours. E.g., Schuman et al. have proposed a convolutional neural network architecture optimized for classifying small objects [93]. They have trained the network with crawled and self acquired small dataset. The network is utilized in the classification part of the Faster R-CNN architecture. Farhadi et al. have utilized the original Faster R-CNN architecture with the help of dynamic background subtraction and position history to refine detections [31]. Likewise, Saqib et al. have trained the original Faster R-CNN architecture with the challenge dataset [92].

The first thing that attracts attention when looked at the table is the grouping of scores. It implies that algorithm design is more important than specifically tuning it for the problem. Considering that the remaining 16 participants failed to achieve meaningful results, and that the penalty of the setting with limit 6 is very close to the most successful methods, much better than all settings of [92], one can conclude that the method has a reasonably good performance. Huang et al. [49] report that Faster R-CNN [83] based methods have more accurate prediction whereas they are much slower than the single stage counterparts. Considering the motivation of the challenge that requires real-time performance, using stage detectors is more appealing although the penalty score and other performance metrics do not take execution time into account. With this notion in mind, utilizing the method in Chapter 3 is more convenient although penalty scores of algorithms [93] and [31] are better.

Table 5.2: Final score of the algorithms on challenge test video. Adapted from [17].

| **Algorithm** | **Penalty** |
|---|---|
| [93] (setting 1) | 1.0000 |
| [93] (setting 2) | 1.2963 |
| [93] (setting 3) | 2.6347 |
| [31] (unique setting) | 3.1896 |
| **[1] (setting 3)** (Method in Chapter 3) | **7.7556** |
| **[1] (setting 2)** (Method in Chapter 3) | **13.3270** |
| **[1] (setting 1)** (Method in Chapter 3) | **18.8449** |
| [92] (setting 2) | 110.7539 |
| [92] (setting 3) | 124.1251 |
| [92] (setting 1) | 176.8701 |

In addition to the challenge results, penalty metric can be utilized to assess the performance of the artificially created dataset. Since the ground truth information is not provided by the challenge organizers it has been extracted manually to construct the Table 5.3. Ground truth extraction methodology may differ from the original one. This may be the cause of difference in penalty scores for the same configurations. However, the table still provides a meaningful assessment.

Table 5.3: Penalty score comparison of different configurations for detection algorithm. Numbers in parenthesis represent the number of iterations in training.

| **Data/Tolerance** | **0** | **2** | **4** | **6** |
|---|---|---|---|---|
| **Artificial Data (10k)** | 3.7434 | 3.1053 | 2.8879 | 3.1125 |
| **Artificial Data (20k)** | 3.8320 | 3.3108 | 3.0506 | 3.2743 |
| **Challenge Data (10k)** | 7.5589 | 7.3129 | 7.1160 | 7.1219 |
| **Challenge Data (20k)** | 9.4443 | 9.2080 | 10.6418 | 10.9510 |
| **Combined Data (10k)** | 5.1821 | 4.5904 | 3.019 | **1.8689** |

As it can easily be seen from the table, training with the artificial dataset is always better than training with the challenge dataset. Since the latter is too scarce, network overfits to training data. The steady scores for artificial data shows that the resulting algorithm is robust and stable. The most successful configuration is the result of training with combined data when the maximum tolerance is 6. With the scores of artificial data, it can be concluded that the results may be improved by training with artificial data and adapting to the problem domain with the combined or domain specific data.

# CHAPTER 6

# CONCLUSION AND DISCUSSION

Increasing availability and improving autopilot technologies of drones pose important threats especially when they are equipped with cameras or weapons. With this motivation, an end-to-end neural network based drone detection method and a novel object tracking architecture are proposed in this thesis.

The evaluations have shown that the drone detection method is a meaningful choice among the alternatives when both accuracy and execution times are a critical issue. It has also ranked 3rd among 20 participants in the Drone-vs-bird challenge [17]. The subjective evaluation of the method on challenge test video has shown that it is successful in most scenarios except its detecting the bird as drone when the bird is closer to the camera and in specific poses that cannot be easily distinguished from a drone by human eye. Another observation is that when the drone and the bird are too close to each other, the network supposes that the bird is a part of the drone, and outputs a bounding box enclosing both of them. This has created a motivation to process time domain information to increase accuracy. Hence, the study continued with drone tracking as a future work of the former one.

In the second part of the study, a novel single object tracking architecture based on Neural Turing Machines has been proposed. It has been discussed that it can easily be converted to a multiple object tracking architecture with the proposed adaptation method. The analysis of the correspondences between the architecture modules and general multiple object tracking algorithm steps has shown that with a successful training strategy and valuable dataset, it can learn the task of object tracking.

In addition to the detection and tracking methods, the thesis adds an extensive image database, which is intended to be used in deep learning approaches for drone detection problem, to the literature. The dataset is named as METU Drone Dataset. The success of the detection method trained with the dataset has revealed the capabilities of the dataset. Since training deep networks requires immense amount of data with high variance, the results have shown that the dataset is a valuable one for drone detection problem.

The proposed object tracking architecture opens many future directions in the domain of single and multiple object tracking algorithms. The first one is obviously designing a training strategy to make it work. Others may include studying the novel modules of the architecture separately. For instance, the architecture requires memory to store information of different objects in different locations. Kalman module is also another novel part that can be studied individually.

Apart from drone tracking problem, the proposed architecture can be adapted to any other tracking problem although it requires off-line training for the specific domain. The architecture has been designed to be class agnostic; hence, it can track multiple objects from multiple classes at the same time. This, of course, depend on the detector in use and off-line training for motion models. Another marvelous future direction is that the detections can be handled in the tracking network with a new detector module that benefits from the spatial and temporal information in the memory module. This modification converts the network to a black box that gets frames of a video and outputs the trajectories of the objects entering and leaving the scene in a robust manner. Considering all future directions, the proposed architecture widens the horizon for the next studies in visual object tracking.

## REFERENCES


[1] C. Aker and S. Kalkan. Using deep networks for drone detection. *CoRR*, abs/1706.05726, 2017.

[2] S. Akhlaghi, N. Zhou, and Z. Huang. Adaptive adjustment of noise covariance in kalman filter for dynamic state estimation. In *Power & Energy Society General Meeting*, pages 1–5. IEEE, 2017.

[3] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1926–1933. IEEE, 2012.

[4] B. Babenko, M. Yang, and S. Belongie. Visual tracking with online multiple instance learning; 2009.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[6] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[7] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1806–1819, 2011.

[8] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pages 850–865. Springer, 2016.

[9] A. Blake and M. Isard. The condensation algorithm-conditional density propagation and applications to visual tracking. In *Advances in Neural Information Processing Systems*, pages 361–367, 1997.

[10] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2544–2550. IEEE, 2010.

[11] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Robust tracking-by-detection using a detector confidence particle filter. In

*2009 IEEE 12th International Conference on Computer Vision*, pages 1515–1522. IEEE, 2009.

[12] W. Brendel, M. Amer, and S. Todorovic. Multiobject tracking as maximum weight independent set. In *2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1273–1280. IEEE, 2011.

[13] A. A. Butt and R. T. Collins. Multi-target tracking by lagrangian relaxation to min-cost network flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1846–1853, 2013.

[14] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European Conference on Computer Vision*, pages 778–792. Springer, 2010.

[15] D. Casasent. Unified synthetic discriminant function computational formulation. *Applied Optics*, 23(10):1620–1627, 1984.

[16] J. Choi, H. J. Chang, S. Yun, T. Fischer, Y. Demiris, J. Y. Choi, et al. Attentional correlation filter network for adaptive visual tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, page 7, 2017.

[17] A. Coluccia, M. Ghenescu, T. Piatrik, G. D. Cubber, A. Schumann, L. W. Sommer, J. Klatte, T. Schuchert, J. Beyerer, M. Farhadi, R. Amandi, C. Aker, S. Kalkan, M. Saqib, N. Sharma, S. D. Khan, K. Makkah, and M. Blumenstein. Drone-vs-bird detection challenge at ieee avss2017. *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2017.

[18] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[19] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, 2003.

[20] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[21] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems 29*, pages 379–387. 2016.

[22] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[23] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Convolutional features for correlation filter based visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 58–66, 2015.

[24] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4310–4318, 2015.

[25] M. Danelljan, A. Robinson, F. S. Khan, and M. Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *European Conference on Computer Vision*, pages 472–488. Springer, 2016.

[26] P. Del Moral. Non linear filtering: Interacting particle solution. 2:555–580, 03 1996.

[27] P. Del Moral et al. Measure-valued processes and interacting particle systems. application to nonlinear filtering problems. *The Annals of Applied Probability*, 8(2):438–495, 1998.

[28] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[29] G. Duan, H. Ai, S. Cao, and S. Lao. Group tracking: exploring mutual relations for multiple object tracking. In *European Conference on Computer Vision*, pages 129–143. Springer, 2012.

[30] K. Fang, Y. Xiang, and S. Savarese. Recurrent autoregressive networks for online multi-object tracking. *arXiv preprint arXiv:1711.02741*, 2017.

[31] M. Farhadi and R. Amandi. Drone detection using combined motion and shape features. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance*. IEEE, 2017.

[32] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[33] H. K. Galoogahi, A. Fagg, and S. Lucey. Learning background-aware correlation filters for visual tracking. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21–26, 2017.

[34] R. Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[35] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[36] F. Gökçe, G. Üçoluk, E. Şahin, and S. Kalkan. Vision-based detection and distance estimation of micro unmanned aerial vehicles. *Sensors*, 15(9):23805–23846, 2015.

[37] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[39] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[40] C. Gulcehre, S. Chandar, K. Cho, and Y. Bengio. Dynamic neural turing machine with soft and hard addressing schemes. *arXiv preprint arXiv:1607.00036*, 2016.

[41] Q. Guo, W. Feng, C. Zhou, R. Huang, L. Wan, and S. Wang. Learning dynamic siamese network for visual object tracking. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[42] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, volume 15, pages 10–5244. Citeseer, 1988.

[43] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, Sept 2015.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[45] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[46] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[47] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park. Pvanet: Lightweight deep neural networks for real-time object detection. 11 2016.

[48] C. Huang, B. Wu, and R. Nevatia. Robust object tracking by hierarchical association of detection responses. In *European Conference on Computer Vision*, pages 788–801. Springer, 2008.

[49] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 4, 2017.

[50] M. Jiang, C. Deng, Z. Pan, L. Wang, and X. Sun. Multiple object tracking in videos based on lstm and deep reinforcement learning. *Complexity*.

[51] Z. Kalal, K. Mikolajczyk, J. Matas, et al. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409, 2012.

[52] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[53] A. Karphaty. The unreasonable effectiveness of recurrent neural networks, May 2015.

[54] L. Kratz and K. Nishino. Tracking with local spatio-temporal motion patterns in extremely crowded scenes. 2010.

[55] H. J. Kushner. Dynamical equations for optimal nonlinear filtering. *Journal of Differential Equations*, 3:179–190, 1967.

[56] L. Leal-Taixé, C. Canton-Ferrer, and K. Schindler. Learning by tracking: Siamese cnn for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 33–40, 2016.

[57] K. Lenc and A. Vedaldi. R-cnn minus r. In *British Machine Vision Conference*, pages 5.1–5.12. BMVA Press, 2015.

[58] Y. Li, C. Huang, and R. Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. 2009.

[59] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[60] B. Liu, Y. Wang, Y.-W. Tai, and C.-K. Tang. Mavot: Memory-augmented video object tracking. *arXiv preprint arXiv:1711.09414*, 2017.

[61] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37, Cham, 2016. Springer International Publishing.

[62] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[63] D. G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, 1999.

[64] W. Luo, X. Zhao, and T. Kim. Multiple object tracking: A review. *CoRR*, abs/1409.7618, 2014.

[65] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang. Hierarchical convolutional features for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3074–3082, 2015.

[66] A. Mahalanobis, B. V. Kumar, and D. Casasent. Minimum average correlation energy filters. *Applied Optics*, 26(17):3633–3640, 1987.

[67] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004.

[68] L. Mejias, S. McNamara, J. Lai, and J. Ford. Vision-based detection and tracking of aerial targets for uav collision avoidance. In *International Conference on Intelligent Robots*, 2010.

[69] A. Milan, S. H. Rezatofighi, A. R. Dick, I. D. Reid, and K. Schindler. Online multi-target tracking using recurrent neural networks. In *Association for the Advancement of Artificial Intelligence*, volume 2, page 4, 2017.

[70] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):58–72, 2014.

[71] D. Mitzel and B. Leibe. Real-time multi-person tracking with detector assisted structure propagation. In *2011 IEEE International Conference on Computer Vision Workshops*, pages 974–981. IEEE, 2011.

[72] M. C. Mozer. A focused backpropagation algorithm for temporal. *Backpropagation: Theory, Architectures, and Applications*, page 137, 1995.

[73] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4293–4302, 2016.

[74] G. Ning, Z. Zhang, C. Huang, X. Ren, H. Wang, C. Cai, and Z. He. Spatially supervised recurrent convolutional neural networks for visual object tracking. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[75] C. Olah. Understanding lstm networks, Aug 2015.

[76] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *Conference on Neural Information Processing Systems Workshops*, 2017.

[77] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1201–1208. IEEE, 2011.

[78] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, J. Lim, and M.-H. Yang. Hedged deep tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4303–4311, 2016.

[79] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[80] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[81] S. Reed and N. De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[82] D. Reid et al. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[83] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, 2015.

[84] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

[85] M. Rodriguez, J. Sivic, I. Laptev, and J.-Y. Audibert. Data-driven crowd analysis in videos. In *13th International Conference on Computer Vision*, pages 1235–1242. IEEE, 2011.

[86] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1-3):125–141, 2008.

[87] A. Rozantsev, V. Lepetit, and P. Fua. Detecting Flying Objects using a Single Moving Camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:879 – 892, 2017.

[88] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *IEEE International Conference on Computer Vision*, pages 2564–2571. IEEE, 2011.

[89] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[90] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[91] A. Sadeghian, A. Alahi, and S. Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. *arXiv preprint arXiv:1701.01909*, 4(5):6, 2017.

[92] M. Saqib, S. D. Khan, N. Sharma, and M. Blumenstein. A study on detecting drones using deep convolutional neural networks. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 1–5. IEEE, 2017.

[93] A. Schumann, L. Sommer, J. Klatte, T. Schuchert, and J. Beyerer. Deep cross-domain flying object classification for robust uav detection. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 1–6. IEEE, 2017.

[94] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[95] J. Shi and C. Tomasi. Good features to track. Technical report, Cornell University, 1993.

[96] X. Shi, H. Ling, J. Xing, and W. Hu. Multi-target tracking by rank-1 tensor approximation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2387–2394, 2013.

[97] H. T. Siegelmann. Computation beyond the turing limit. *Science*, 268(5210):545–548, 1995.

[98] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[99] J. Sivic, A. Zisserman, et al. Video google: A text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision*, 2003.

[100] J. Son, M. Baek, M. Cho, and B. Han. Multi-object tracking with quadruplet convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5620–5629, 2017.

[101] R. Tao, E. Gavves, and A. W. Smeulders. Siamese instance search for tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1420–1429, 2016.

[102] Z. Teng, J. Xing, Q. Wang, C. Lang, S. Feng, Y. Jin, et al. Robust object tracking based on temporal and spatial deep networks. In *IEEE International Conference on Computer Vision*, volume 2, 2017.

[103] M. Trajković and M. Hedley. Fast corner detection. *Image and Vision Computing*, 16(2):75–87, 1998.

[104] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[105] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001.*, volume 1, pages I–I. IEEE, 2001.

[106] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3119–3127, 2015.

[107] L. Wang, W. Ouyang, X. Wang, and H. Lu. Stct: Sequentially training convolutional networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1373–1381, 2016.

[108] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339 – 356, 1988.

[109] J. Xing, H. Ai, and S. Lao. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1200–1207. IEEE, 2009.

[110] B. Yang and R. Nevatia. Multi-target tracking by online learning of non-linear motion patterns and robust appearance models. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1918–1925. IEEE, 2012.

[111] M. Yang, F. Lv, W. Xu, and Y. Gong. Detection driven adaptive multi-cue integration for multiple human tracking. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1554–1561. IEEE, 2009.

[112] R. Yoshihashi, T. T. Trinh, R. Kawakami, S. You, M. Iida, and T. Naemura. Differentiating objects by motion: Joint detection and tracking of small flying objects. *arXiv preprint arXiv:1709.04666*, 2017.

[113] W. Zaremba and I. Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[114] W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015.

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