Multi-Template Temporal Siamese Network for Visual Object Tracking

by

Ali Sekhavati

A thesis submitted in partial fulfillment of the requirements for the

Master's degree in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

University of Ottawa

Jan 2023

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. The reprint permission is obtained through the publishers.

ABSTRACT

Visual object tracking is the task of giving a unique ID to an object in a video frame, understanding whether it is present or not in a current frame and if it is present, precisely localizing its position. There are numerous challenges in object tracking, such as change of illumination, partial or full occlusion, change of target appearance, blurring caused by camera movement, presence of similar objects to the target, changes in video image quality through time, etc. Due to these challenges, traditional computer vision techniques cannot perform high-quality tracking, especially for long-term tracking.

Almost all the state-of-the-art methods in object tracking use artificial intelligence nowadays, and more specifically, Convolutional Neural Networks. In this work, we present a Siamese based tracker which is different from previous works in two ways.

Firstly, most of the Siamese based trackers takes the target in the first frame as the ground truth. Despite the success of such methods in previous years, it does not guarantee robust tracking as it cannot handle many of the challenges causing change in target appearance, such as blurring caused by camera movement, occlusion, pose variation, etc. In this work, while keeping the first frame as a template, we add five other additional templates that are dynamically updated and replaced considering target classification score in different frames. Diversity, similarity and recency are criteria to choose the members of the bag. We call it as a bag of dynamic templates.

Secondly, many Siamese based trackers are vulnerable to mistakenly tracking another similar looking object instead of the intended target. Many researchers proposed computationally expensive approaches, such as tracking all the distractors and the given target and discriminate them in every frame. In this work, we propose an approach to handle this issue by estimate the next frame position by using the target's bounding box coordinates in previous

frames. We use temporal network with past history of several previous frames, measure classification scores of candidates considering templates in the bag of dynamic templates and use tracker sequential confidence value which shows how confident the tracker has been in previous frames. We call it as robustifier that prevents the tracker from continuously switching between the target and possible distractors with this hypothesis in mind.

Extensive experiments on OTB 50, OTB 100 and UAV20L datasets demonstrate the superiority of our work over the state-of-the-art methods.

Acknowledgment

First of all, I would like to thank my family, especially my parents for encouraging me to do research and for all the love and support I received from them. Without their help it would be impossible for me to reach here.

I would like to express my sincere gratitude to my supervisor, Professor WonSook Lee, who helped me with her advice and support during my studies in master's degree. I learned a lot about state-of-the-art research on computer vision in LIII++ seminars. I could not ask for a better supervisor. Her knowledge, kind guidance and supports greatly helped me during this research.

Last but not least, I would like to thank my previous teachers, friends, and teammates with whom I had the pleasure of meeting or collaborating.

Table of Contents

List of Figures

List of Tables

Epigraph

"An expert is a person who has made all the mistakes that can be made in a very narrow field."

—Neils Bohr

Abbreviations

ANN: Artificial Neural Network

BCE: Binary Cross-Entropy

BiFPN: Bi-directional Feature Pyramid Network

CE: Cross-Entropy

CNN: Convolutional Neural Network

DL: Deep Learning

FPN: Feature Pyramid Network

FFT: Fast Fourier Transform

FPS: Frames Per Second

HOG: Histogram of Oriented Gradients

IFFT: Inverse Fast Fourier Transform

MLP: Multi-Layer Perceptron

MTTSiam: Multi-Template Temporal Siamese network

NLP: Natural Language Processing

PSR: Peak to Sidelobe Ratio

ReLU: Rectified Linear Unit

RMSE: Root Mean Square Error

RoI: Region of Interest

RPN: Region Proposal Network

SGD: Stochastic Gradient Descent

Chapter 1. Introduction

Visual object tracking is the task of localizing a given object in different video frames. It can include object detection, segmentation, classification, etc. An important factor that makes tracking difficult is that targets can be any arbitrary objects, such as a plastic bag. Tracking can be done either using deep neural networks or using other means, such as filter optimization, key point tracking, k-means clustering, etc. Since 2010 when Bolme et al. [1] introduced the MOSSE tracker until 2016, state of the art methods were heavily relying on adaptive correlation filters. The idea of using a filter was successful in terms of speed and addressing changes to the target during tracking. However, the problem with using a filter made from a sequence of images and finding the target by a single correlation operation is that it is not able to properly

learn target features. Another method that can be used for object tracking is learning target features using a deep neural networks and using the learned features to distinguish the target from the background. In recent years, computer vision and object tracking has been greatly enhanced by deep learning approaches. For instance, in VOT2022 single object tracking challenge, among 9 trackers proposed by different authors, all of them use CNN (Convolutional Neural Network) for feature extraction and four of them use transformer architecture as well.

When tracking using deep learning methos, the network used for tracking might not be already trained to identify that specific object. For handling this type of issue, researchers use two major architectures for making their models: Siamese-based trackers and filter-based trackers.

A Siamese network is a type of network specialized in recognition with one shot learning. A Siamese network is technically two identical networks with identical weights, layers, parameters, etc. Two images can be fed to Siamese networks and with correct training, if the two images belong to the same object the outputs are quite similar, otherwise different. One shot learning is a problem in which classification is done using only one data. For instance, in face recognition if a model is able to recognize a face using only one other image, that model is performing one-shot learning. Early Siamese networks were mainly applied for face or signature recognition. For training a Siamese network traditionally, at least three images are required: A template image, a positive sample and a negative sample. During training, the Siamese network must maximize the similarity between the template and the positive sample while minimizing the similarity between the template and the negative sample. For face recognition, the template can be considered someone's facial image. The positive sample can be their facial image from another angle, and the negative sample can be another person's face. For object tracking, most prior works take the target image given in the first frame i.e., ground truth information, as the template, and the target in future frames are positive samples. Negative

samples can be background information or objects similar to the target. It is proved that using similar objects as negative samples can enhance the ability of the network to identify the target better.

The filter-based trackers mostly rely on a convolutional network to extract features from the target, and then they produce a filter of the target's features. In every frame, they update the filter to better address challenges during the tracking, such as new changes in target appearance or changes in illumination. Despite their superiority in short-term tracking, they cannot handle long occlusions and disocclusions.

The early famous filter-based trackers were first designed in 2010 using only grayscale image pixels. They would make a template of the target in the first frame, apply the template's correlation over the entire next image, and get a score map. The highest score on the resulting score map would be where the target most probably is. They would also use fast Fourier transform (FFT) to perform a single multiplication instead of the correlation operation. After processing every frame, the template should get updated to minimize the score for the background and maximize it for the real target. Different approaches for filter design were used, such as MOSSE [1], which minimizes the sum of squared error. This approach resulted in robust trackers that state of the art methods used before 2016.

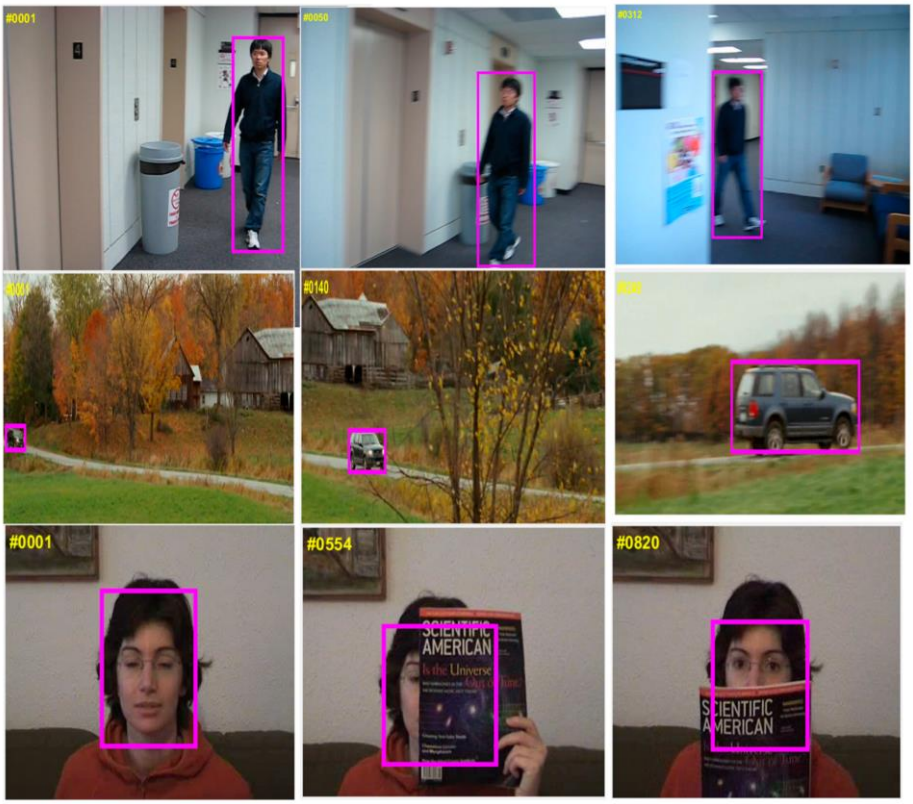

Figure 1-1: Some of the challenges in visual object tracking, including camera motion in the 1st row images, change of view in the 2nd row images, and partial and full occlusion in the 3rd row Figures. The images are taken from the OTB 100 dataset [9].

Considering video length and challenges in every video for visual object tracking, this task can be categorized into short-term object tracking and long-term object tracking.

1.1 Short-term object tracking

Short-term object tracking, which is relatively not as difficult as long-term object tracking, is the task of modeling a target appearance and precisely localizing it in every video frame. Videos in short-term object tracking are usually a few seconds long. Although there are not a lot of frames in short-term tracking datasets videos, there are many factors that make it complicated. Challenges in short-term tracking mainly include partial occlusion, change of

view, camera motion, presence of possible distractors, change of illumination, and in many cases, a mixture of two or more of them.

For tackling challenges in short-term tracking, a tracker must first be able to adapt itself to the changes. This is why most successful works in short-term tracking are filter-based methods, because filters can be updated during tracking. For instance,

1.2 Long-term object tracking

Long-term object tracking is slightly more challenging than the short-term one. Videos in long-term tracking are usually a few minutes long. In addition to challenges in short-term tracking, long disappearance and reappearance of the target are also very common in datasets for this type of tracking. Its challenging nature requires more computational power and it is closer to real-life applications. Long-term trackers need to be able to reidentify given targets after long disappearances. They should also report their absence when they are not visible in given frames. Since target appearance changes during tracking, they need to be able to address those changes as well. However, in practice many long-term trackers only rely on target features extracted from the first frame to avoid wrong updates, as have a wrong model is very undesired in long videos.

1.3 Motivation

This work utilizes Siamese architecture for robust tracking and is based on the SiamRPN++ architecture proposed by Li et al. [8]. The architecture they designed comprises two essential parts: A backbone network that extracts features from the template and search image, and three region proposal heads, each consisting of a classification branch and a regression branch. Each region proposal head is responsible for proposing regions where the target might be found. Despite its well-designed architecture, it has two major problems:

Firstly, it only uses the target appearance in the first frame to localize it in future frames. This approach is not capable of addressing target appearance changes, blurring caused by camera movement, change of illumination, etc. in later frames. Filter-based trackers are better at capturing recent target changes, but they are not suitable for long-term tracking.

Secondly, many detection-based trackers are prune to mistakenly track an object similar to the target, also known as a distractor, instead of a given target. Since Siamese trackers are detection-based algorithms, in many cases they either track a distractor or continuously switch between the given target and different distractors.

Observing these two issues with other trackers, we decided to propose a long-term tracker that can handle the problems with little change in computation complexity. We propose a bag of dynamic templates which greatly increases tracker success and precision rates by solving the first problem. We also propose a candidate selection algorithm which understands how confident the tracker has been working in previous frames, predict target bounding box coordinates and select the most likely candidate considering all these three concepts. In the next section we will review the architecture of this tracker and how it performs each of these tasks.

1.4 Overview of the Pipeline

We extend the number of templates to 6 so that the model can learn different possible shapes of the target, focusing on recent ones. It is critical to keep the first frame information and never replace it with newer templates, as long-term tracking is quite challenging and might lead trackers to update their models online with wrong information. Each of those five templates is responsible for containing the target appearance features within a specific range of similarity. For instance, one template is responsible for containing the newest target features whenever the classification score is above 99%. Another template should contain target

features when the classification score is between 80% and 85%, which is considered as a diverse appearance of the target.

We also trained a temporal network on the GOT10K dataset for learning the path information and predicting where the target might be in the next frame. As described in chapter 3, we tried several different architectures until choosing the best one.

Finally, we proposed a candidate selection algorithm that combines all the previously mentioned information and gives a reliability score to those proposed candidates. The reliability score is the milestone used in this tracker to determine whether the candidate is the actual given target or not.

In this work, we improved the SiamRPN++ tracker to achieve better results with fewer computations. When using AlexNet [19] as the backbone for MTTSiam, it runs 4x faster than the SiamRPN++ when it uses the ResNet 50 as the backbone and the precision obtained on the OTB 100 dataset for both of the models is almost the same. Moreover, the tracking results on the UAV20L dataset proves that this tracker achieves better success rate and precision than state-of-the-art methods by a large margin. More details are provided in chapter 4.

To track an object in a video the target appearance is provided to us in the first frame. MTTSiam takes the target image in that frame and feeds it into the backbone. The algorithm has now the target feature template in the most important one of those six templates. Then, it takes the search region for every frame and feeds it into the backbone. In the RPN heads, it performs a depth wise cross-correlation between features of the six templates and the search region features to produce the classification and regression feature maps for each of the produced anchors. After that, a weighted average of all the produced feature maps is obtained to produce the final feature maps. In this step, the top ten candidates with the highest classification scores are saved for further processing in the second stage. The classification

branch of RPN heads performs binary classification, showing whether the features in an anchor belong to the target or not. Thus, the classification score is a measure of similarity.

The next step is using the temporal network for bounding box coordinates prediction. It takes the target coordinates in the previous four frames as the input, and produces the target bounding box coordinate in the current frame as the output. The candidate selection algorithm first computes how far each candidate is from the predicted position. Then it uses the distance error and combines it with the classification score and a sequence confidence value to produce the final reliability score. If the distance error is high, negatively affects the classification score; if it is low, its effect is positive. The sequence confidence value updates itself in every frame, and determines how confident the tracker has been in previous frames; if the sequence confidence value is low, the candidate selection algorithm decreases the effect of path information over classification the score; if the value is high, the candidate selection algorithm increases the effect of path information.

There are two important problems with state-of-the-art Siamese based trackers. Most of the Siamese based trackers rely on the target appearance in the first frame, which reduces their ability to handle target appearance changes during tracking. Our bag of dynamic templates is responsible for handling this issue. The other problem is that most state-of-the-art trackers are highly based on detection, which leads to the confusion of the tracker between the real target and distractors with similar appearance. They can repetitively switch between a target and similar objects. Our temporal network and candidate selection algorithm tries to solve this problem by estimating target location.

1.5 Contributions

There are three technical contributions in this work:

- A bag of dynamic multi-templates is proposed with a new approach for using recent information of the target.
- We proposed a new temporal network for predicting the target bounding box coordinates given the coordinates in the previous four frames.
- We propose a candidate selection algorithm that produces a reliability score for each candidate. It considers the classification scores, candidate coordinates, and tracker sequence confidence value.

1.6 Thesis Structure

The rest of this thesis is made of the following sections:

- Chapter 2 briefly reviews how neural networks and deep learning work, computer vision techniques, Siamese networks and correlation-based trackers, and other related works.
- Chapter 3 contains the methodology of this work, the goal for each part of the pipeline and an explanation over the parameters and equations used.
- Chapter 4 covers the results obtained by this tracker over the OTB 100 dataset and qualitative results with common phone cameras, and a comparison with the SiamRPN++ tracker.
- Chapter 5 summarizes the proposed method, our contributions, and potential future works.

Chapter 2. Literature Review

This chapter covers a review of other related methods, such as different trackers and other techniques used in this work. Section 2.1 reviews some background information on deep learning. After that, section 2.2 reviews some critical background information in traditional computer vision for object tracking. Finally, 2.3 covers a brief history of object detection methods, as it plays a crucial role in object tracking, and a review of some modern essential works in object tracking that lead to this work.

2.1 Deep Learning Background

This section provides a brief review of different deep learning components.

2.1.1 Artificial Neural Networks

A neural network consists of different layers, each containing several neurons. The input to each neuron is either the primary input to the network or the output of other neurons. Each neuron multiplies its inputs by specific learnable parameters, named weights of that neuron. Then it adds the result of those multiplications with another learnable parameter named bias. Finally, it applies a nonlinear function, called the activation function, to its output and feeds its output into the neurons in the next layer. Figure 2.1 is designed to understand better how each neuron in an artificial neural network works.

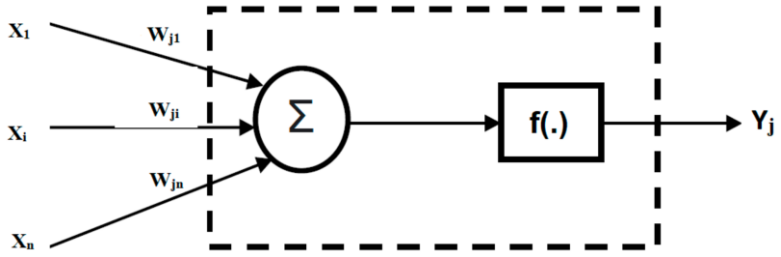

Figure 2-1: Overview of how a single neuron works. It sums the result of multiplications described in the text and adds the result to an arbitrary learnable parameter named bias. Then nonlinear function f(.) is used to prevent the network to remain linear. [10]

Artificial Neural Networks, or in short ANNs, are designed to learn how to map input x to output y using the learnable parameters in neurons. Figure 2.2 shows an overview of the structure of a neural network. It shows a fully connected network. A fully connected network is a network in which the output of all neurons in one layer is connected to the input of all neurons in the next layer.

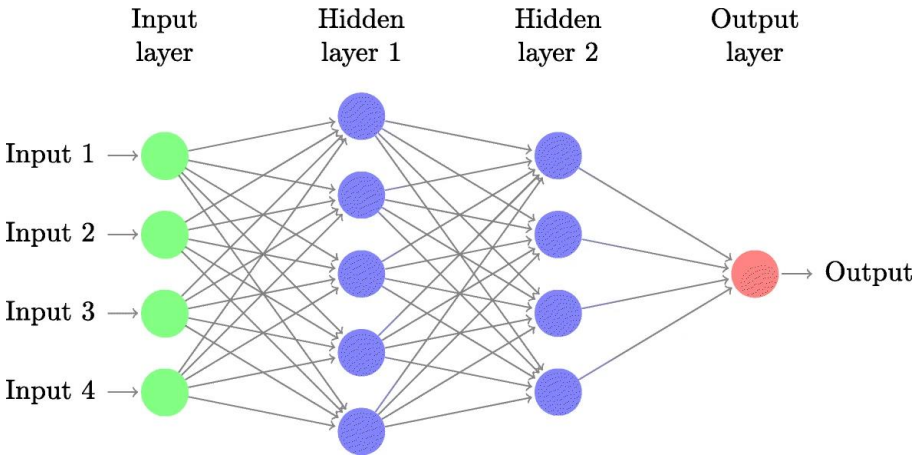

Figure 2-2:An overview of a neural network. Source: [11]

As shown in Figure 2.2, the network takes an input, such as image pixel values. It multiplies all the input values to the randomly initialized weights of different neurons in the input layer. Then, every neuron sums the result of its multiplications and adds a random bias value to the result. After that, it performs a nonlinear operation on the result and generates its output. More details about nonlinear operations for neurons known as activation functions can be found in section 2.1.2. After all the neurons in all the layers perform their operations, the last layer output is considered the network output. In supervised learning, which is the type of machine learning that we use in this work, the output is compared with the desired output. For example, suppose we want to train a network that can classify whether an input image is a cat or dog. Firstly, we feed the pixel values of a cat image to the network. Only one neuron is enough to determine the input image in the output layer. We encode the labels as 0 and 1, meaning that if the output of the last neuron is 0, it means that it is a cat image, and if it is 1, it means that the image has a dog in itself. Next, we compare the network output with the desired output, i.e., the label using a loss function. More details about loss functions are available in section 2.1.3. Finally, the comparison result is fed back into the network using a process named backpropagation, which is described in section 2.1.4. Through that process, network weights

and biases are updated so that its output gets closer to 0, which is the cat label. After repeating this process for many input images, then the network can learn how to distinguish a cat image from a dog image.

It is worth noting that the training and test data should be from the same distribution for the network to perform well. For instance, if a network is trained using some cat and dog images taken from very high-quality cameras, it will not perform well when it faces low-quality images taken from old mobile phones. This is one of the reasons that using more training data will most probably improve the model results unless the model capacity is low. Model capacity is considered low when increasing the number of layers or neurons improves the results.

2.1.2 Activation Functions

Activation functions are nonlinear functions applied to the output of neurons to add nonlinearity to networks. Every neuron in hidden layers should have an activation function because the whole network will only be a set of multiplication and summation operations performed on inputs without them. Since multiplication and summation are linear operations, the whole network will perform like a linear operation without activation functions, such as a single multiplication or a type of transform done using only one layer. In other words, a network with many hidden layers without activation functions is similar to a network with one single layer.

Many different activation functions can be used in every neuron. Despite not being quite intuitive, recent studies by Dahl et al. [12] suggest the significant influence of the Rectified Linear Unit on neural networks when used in hidden layers. A Rectified Linear Unit, or ReLU in short, is a nonlinear transform that zeros out all the negative inputs and returns non-negative values unchanged. Equation (2.1) shows the mathematical formulation of ReLU:

$$f(z) = (z, 0) \tag{2.1}$$

where z is the input, and f denotes the ReLU activation function.

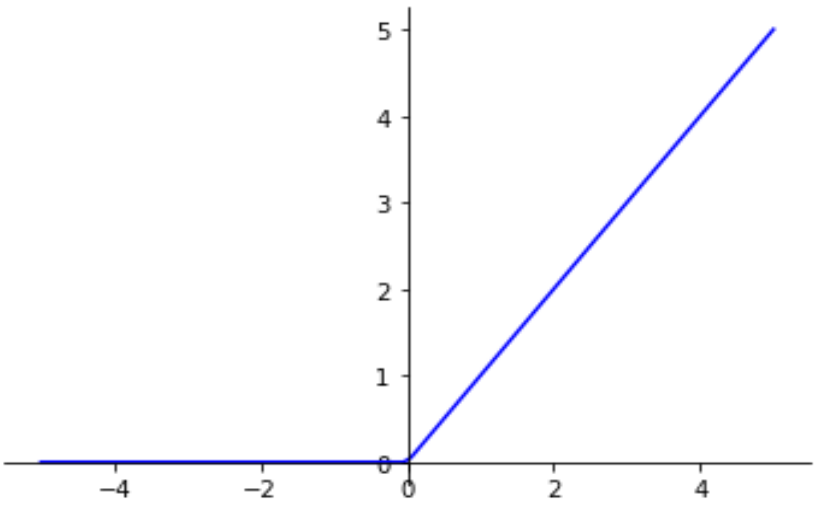

Figure 2-3: ReLU activation function. Source: [13]

In some applications, it might not be the best idea to zero out all negative numbers in hidden layers, which leads to the invention of a leaky ReLU activation function. Figure 2.4 demonstrates an approximation of leaky ReLU:

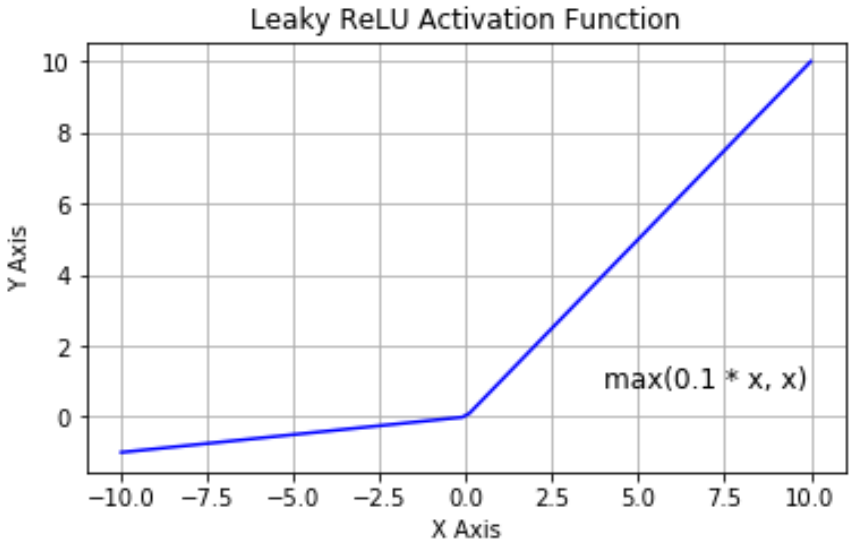

Figure 2-4: An approximation of leaky ReLU activation function. Its difference from the ReLU function is that it does not remove negative values. [14]

$$f(z) = (0.01z, z) \tag{2.2}$$

where z is the input, and f denotes the leaky ReLU activation function.

Although ReLU and leaky ReLU are popular choices for activation functions in neurons in hidden layers, they are not very common for output layer neurons. For instance, when the task is logistic classification, and you want to train a network to learn to which class a particular input belongs, it is better to choose an activation function that keeps the output between 0 and 1. A Sigmoid activation function is a good example of a nonlinear operation that always keeps the output between zero and one. Figure 2-5 shows how sigmoid changes its input.

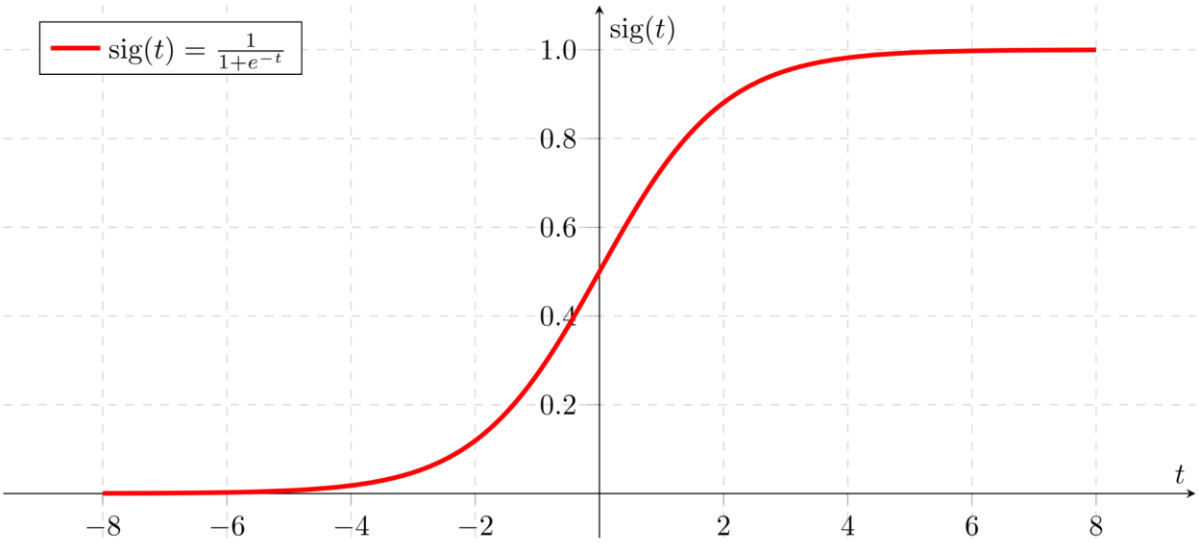

Figure 2-5: Sigmoid activation function with its formula. Source: [15]

Sigmoid is a common choice among deep learning practitioners for the output layer when they want to keep the output between zero and one. It is better to choose other activation functions for some regression tasks where the output needs to be numbers higher than one or below zero.

There are so many different activation functions popular among researchers who work on neural networks and we covered a brief overview of a few of them. Some of them, such as Tanh, keep the output between zero and one. Sometimes, it is best not to choose an activation function for the final layer neurons. Knowing which activation function works best is entirely dependent on the task and network architecture.

2.1.3 Loss Functions

In supervised learning, after the network makes a particular prediction for a given input, it is time to compare the network prediction with actual labels. Using this comparison, we can have a numerical value showing us how different the network output is from the expected value.

Depending on the task, the loss function can have many different shapes. This study used the L1 loss to train the temporal network. What L1 loss does is that it computes the sum of absolute errors for a single prediction. For instance, the temporal network trained in this study predicts four numbers to define a single bounding box coordination in the next frame. The L1 loss computes the difference between each prediction and its ground-truth value and sums up all four error values. The formula for L1 loss is given in equation (2.3)

$$L1_{Loss\ Function} = \sum_{i=1}^{n} \left| y_{true} - y_{predicted} \right| \tag{2.3}$$

where i is the number of predictions, y_{true} is the true label and $y_{predicted}$ is model prediction.

Even though the L1 loss is a popular choice for regression, it is not the only one. There are many different loss functions available for different tasks. Some of them are logarithmic loss functions. Equation (2.4) shows the Binary Cross-Entropy Loss Function, which is a logarithmic loss and a common choice for 2 class classifications:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{2.4}$$

where N is the output size, q is the input, y_i is the true label for sample i and $p(y_i)$ is the predicted value for sample i by the model.

Binary Cross-Entropy, or in short BCE, is also the loss function used for training the classification branch of the SiamRPN tracker [16], which is a work similar to the baseline of this study.

2.1.4 Backpropagation

Finally, after the network has made a prediction and we measured how far the prediction is from the ground truth, it is time to change the weights and biases of the network so that it makes better predictions next time. Gradient Descent is a basic algorithm used for backpropagation. Firstly, it computes the derivative of every parameter concerning the output. Then, it multiplies the derivative with a hyperparameter named learning rate, usually a number less than 1. Then for every parameter in a neural network, this value gets subtracted from its corresponding parameter. This simple process leads the network towards minimizing obtained errors. Figure 2.6 illustrates how gradient descent works with an example of a ball as network parameters. After each network prediction, the parameters get updated to lead the network towards having less error in the output.

If the whole dataset is fed into the network and then the network performs backpropagation once, it is known as gradient descent. If we feed the data one by one in the dataset to the network, it is known as stochastic gradient descent. If we make a group of inputs, also known as a batch, and feed it into the network, it is known as mini-batch gradient descent. Nowadays, usually, it is impossible to feed all the datasets at once to the network due to a lack of memory and abundance of training data in most cases. It also is not efficient to train the network on the data one by one, which is done in stochastic gradient descent, also known as SGD. As a result, most deep learning practitioners train their networks using mini-batches. It was also used for training the temporal network in this study. This is the basis of how all the networks including mulit-layer perceptrons, CNNs, transformer, etc. are trained.

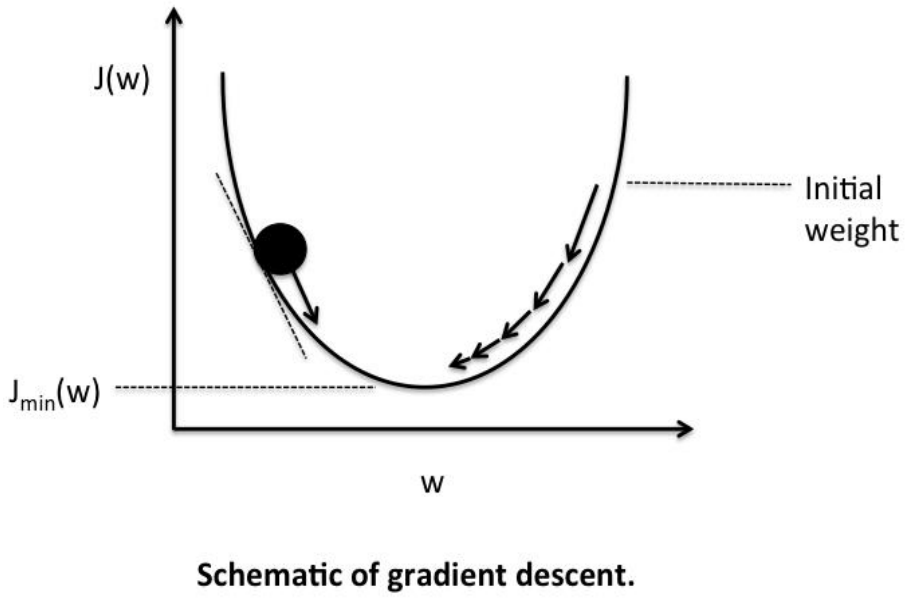

Figure 2-6: A schematic of gradient descent leading parameters towards minimizing the error. The curve shows the loss function. [17]

2.1.5 Convolutional Neural Networks

Convolutional Neural Networks, or CNNs, are the most common type of network when working with images [18]. In convolutional networks, convolution filters with randomly initialized values are applied to the input image and after applying a non-linear operation, such as ReLU, produce feature maps. Then, other convolutional filters are applied to the feature map and produce deeper features. In the end, after obtaining deep features the output is fed into a multi-layer perceptron to perform down-stream tasks, such as image classification, object detection, etc. The process of training a CNN is the same as training other types of networks, explained in section 2.1.4. Alex Krizhevsky et al. [19] made the famous AlexNet network one of the earliest CNNs that used the ReLU activation function and dropout. That network won

the ImageNet 2012 competition and motivated many others to use CNNs. When working with images, there are many differences between multi-layer perceptron (MLP) networks and CNNs. For instance, unlike fully connected networks that focus on global features, this network is more suitable for extracting local features. To combine local features extracted by convolutional neural networks globally, many researchers add fully connected layers at the end of the network. In this way, the network has access to local and global features with so much less computations required.

One can use only fully connected layers for images without using convolutional layers initially. Still, since all neurons in every layer are connected to the ones on the next layer, it will not be computationally efficient. In many studies in computer vision and natural language processing (NLP), convolutional neural networks are used as backbones for feature extraction; and many researchers made their pipelines on top of backbone features as it gives better results.

There are different types of convolutional neural networks. 1D CNN is used for one-dimensional inputs, such as words in a text. 2D CNN can be used for voice recognition or gray scale images. More details about CNNs and filters are available in Figure 2.7.

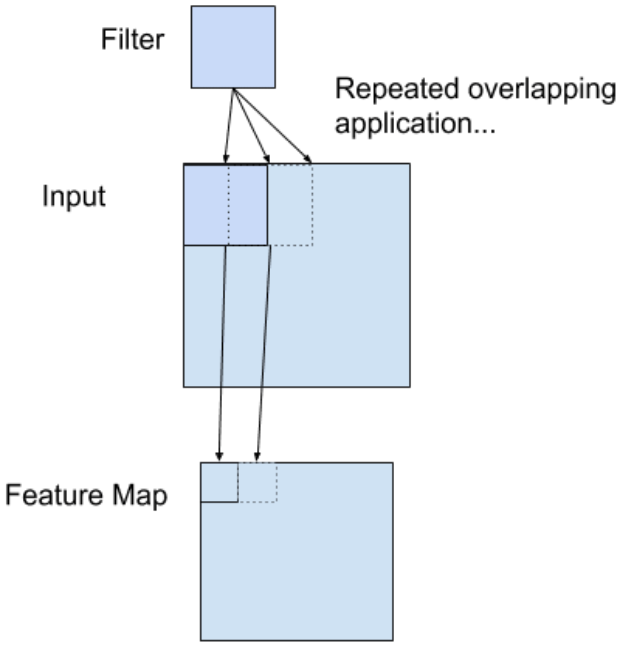

Figure 2-7: How filters in 2-dimensional convolutional neural networks work. Given a 2D input like an image, a filter with learnable parameters cross-correlates the input image and produces the output feature map. The output size is typically smaller. Padding is o

Some famous and commonly used techniques used in CNNs highly enhance their performance. Researchers and deep learning practitioners must learn about those techniques to design better networks. In the following parts, we review two of the most important ones: padding and pooling.

2.1.5.1 Padding

One famous approach for keeping input size unchanged after performing convolution is padding. This technique is common among both deep learning scientists and image processing researchers. Padding refers to simply increasing the size of the image or feature map before applying convolution or cross-correlation to it. There are different types of padding. In a popular type of padding, pixels or features with a value of zero are added to the sides of the inputs. This is known as zeros padding. In another form of padding, the adjacent pixels or

features are copied and added to the sides of the input. Figure 2.8 illustrates how different padding types can increase the input size.

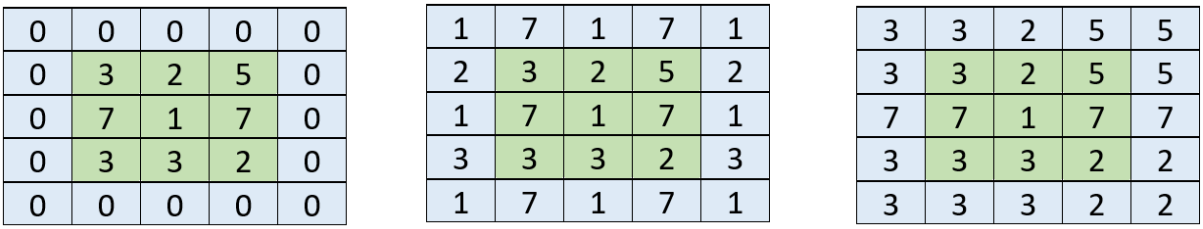

Figure 2-8: Three different padding types. The right image shows replication padding. The image in the middle illustrates replication padding, and the left image is an example of zero padding. [21]

2.1.5.2 Pooling

One of the things that make convolutional neural networks more efficient than fully connected networks is pooling layers. Pooling layers are used for reducing features in a feature map. There are different types of pooling. Max pooling is a commonly used type of pooling used by many researchers [23]. Suppose that there is a 10x10 feature map, and a 2x2 max pooling kernel is used. This pooling kernel reduces the feature map size to 5x5 by keeping the maximum value in every 2x2 neighbor.

Another famous type of pooling is the average pooling. In that type, as the pooling kernel slides over the input, it replaces those values in the kernel with their average.

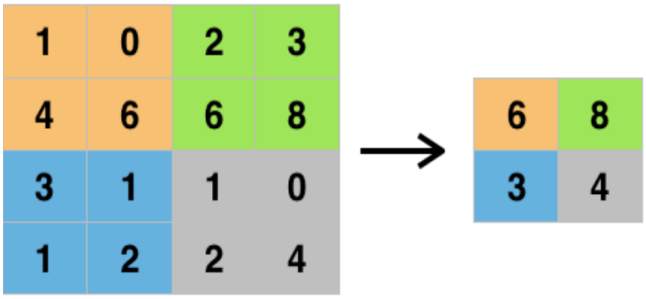

Figure 2-9: Max-pooling can decrease feature map size and network computation. [23]

2.1.6 Overfitting

One of the problems with deep learning algorithms is that they are quite prone to overfitting. Overfitting is a term used when the model performs very well on the training set, but poorly on the test set. There are many ways to handle this issue. For instance, adding more samples to the training set is helpful in many cases. There are some other techniques used for this purpose, such as data augmentation techniques, L1 and L2 regularization and dropout. We will have a brief look at them in the following sections.

2.1.6.1 Regularization

One of the main things that cause a network to overfit is its extremely high capacity. In other words, when the task is relatively simple, too many layers and neurons in a network do not help and cause overfitting. L1 regularization is a technique to reduce the effect of unnecessary parameters, such as the weights of a network. In that technique, the sum of absolute values of weights is multiplied by a coefficient and added to the loss function. In this way, as the network tries to minimize its error in the cost function, it will also try to minimize its parameters' sum of absolute values. Equation (2.5) shows the final loss function when the original loss is Mean Squared Error, and L1 regularization is applied

$$J(\theta) = MSE(\theta) + \lambda \sum_{i=1}^{n} |\theta_i| \quad (2.5)$$

where J is the cost function, θ is the network parameter and λ is a coefficient determining the effect of regularization over the mean square error of network predictions with ground truth labels.

2.1.6.2 L2 Regularization

Similar to L1 regularization, L2 regularization is used for reducing the weights of networks. It computes the squared value of each of the network parameters, sums them and after multiplying with a coefficient, adds it to the loss value. Equation (2.6) shows L2 regularization added to the loss function, where θ is the network parameter, and λ is L2 regularization coefficient.

$$loss = loss + \lambda \sum_{i=1}^{n} \theta_i^2 \quad (2.6)$$

2.1.6.3 Data Augmentation

An important technique mainly used by deep learning researchers when working with images is data augmentation. Data augmentation means using different methods to increase the number of training samples without actually adding real-world data to the dataset. There are many different augmentation techniques available when working with pictures. Image rotation is among the most famous augmentation methods. Whether we rotate an image or not, the information in the image does not change; the networks need to learn this as well.

Another critical method is image flipping. We as human beings can understand how similar information that flipped images and the original ones contain is; but it is not as simple for neural networks. It is essential to add them to the training set to train more robust networks.

Two other important ways to produce more images for training models are random cropping and color distortion. In recent studies [25], the importance of combining cropping and color distortion is shown to have a significant effect on training an unsupervised model to learn image features.

There are many other ways to perform data augmentation on images. Figure 2.10 shows some of these techniques. Cutting out random parts of images, blurring and resizing are among other approaches for image augmentation. Sometimes it is better to combine different augmentations instead of using only one type of augmentation at once, similar to [25].

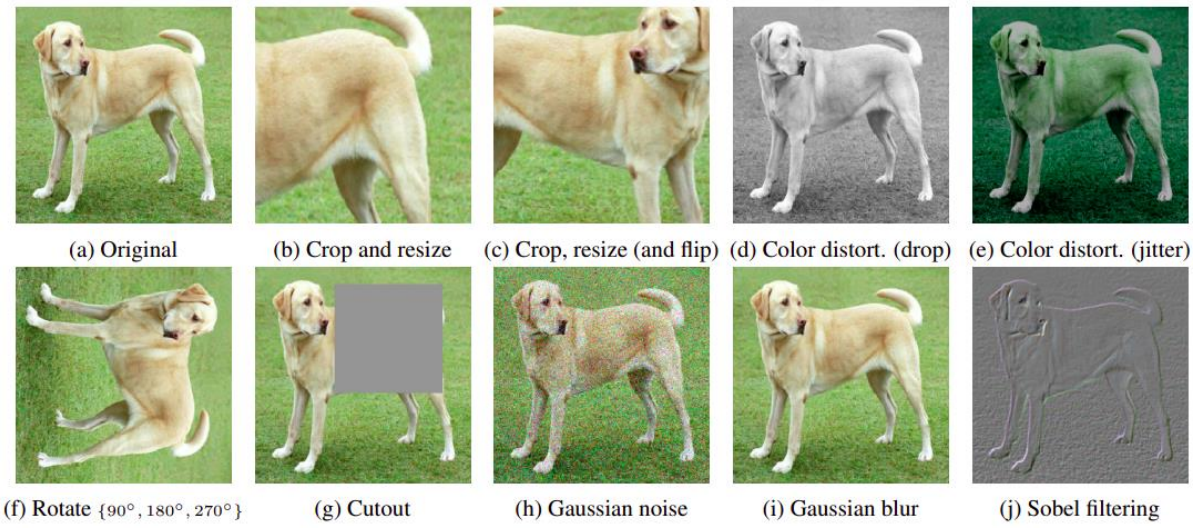

Figure 2-10: Different image augmentations used in State-of-the-Art papers. [25]

2.1.6.4 Dropout

Dropout is another well-known technique used for regularization. The idea behind dropout is quite simple and intuitive. During the training process of different networks,

sometimes the network relies on some specific neurons to make the predictions, while ignoring the rest. This phenomenon reduces network capacity and it prevents the model from generalization. One proposed solution is using dropout regularization. In this method, some of the neurons in a network are randomly selected and deactivated so that the network makes its prediction without them. This can considerably reduce the effect of networks relying only on a few neurons. Moreover, this can add noise to the network and make the network better in generalization. Using dropout is similar to training many different networks with different architectures.

Similar to the idea, implementing dropout in networks is quite simple as well. To implement dropout, we have to randomly deactivate some neurons. For this purpose, first we need to decide the percentage of neurons we want to keep. For instance, suppose during the training every time we want to keep 60% of the neurons and randomly remove 40% of them. This can simply be achieved by defining a threshold for keeping the neurons. Let us call this threshold "keep_prob". In this case, the keep_prob is 60%. Then during the training, we generate random numbers between zero and one for each of the network nodes, using a random uniform distribution. If the generated number for a neuron is above the keep_prob threshold, that neuron's weights and bias are temporally multiplied by zero; otherwise, no operation is done on them.

Despite the simplicity of the idea, there are issues that researchers need to be careful about when implementing dropout. One of the issues is that when implementing dropout, the output of each layer decreases, since some of the neurons are not there to produce input to neurons in the next layer. The input to subsequent layer neurons decreases by different numbers each time. However, it is not wrong to approximate that number by a factor of the keep_prob. In other words, if we divide each node output by the keep_prob, the sum of each layer output will be similar to the sum of each layer output before implementing dropout. It is also worth

mentioning that dropout should be done only during the training, and when the network is being tested on the test set all neurons should be active. Figure 2.11 shows the effect of dropout on standard networks.

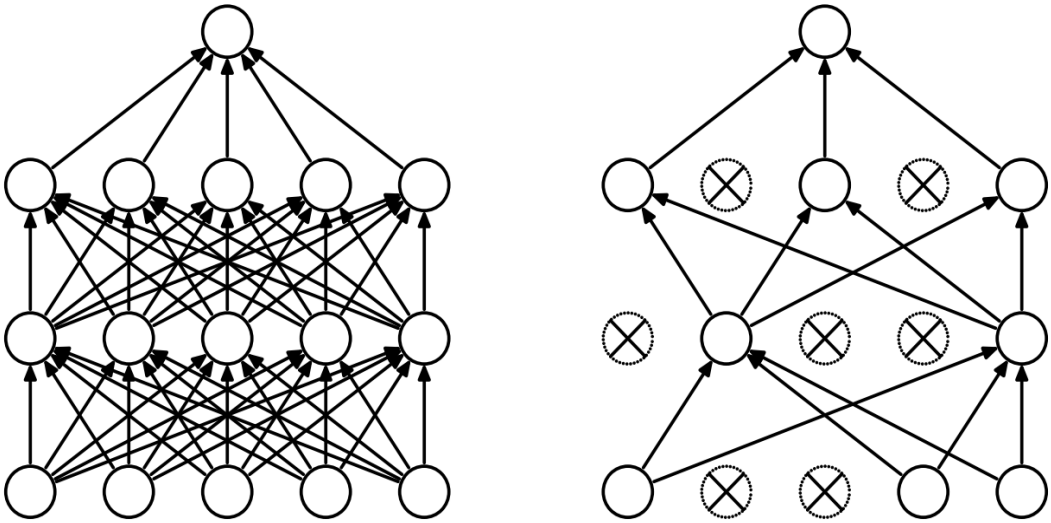

Figure 2-11: How dropout changes the architecture of the network each time in the training process. [26]

2.1.7 Transfer Learning and Use of Pre-Trained Backbones

There is not quite enough data available for training in many deep learning tasks. One of the things that most researchers do is use a model that is already trained for another task and finetune its parameters on their own task. This is known as transfer learning, where the learned model parameters are used and finetuned on another task. For example, the baseline of the SiamRPN++ tracker [8], that is the SiamRPN tracker [16] uses a pretrained AlexNet network [19] that is trained on the ImageNet dataset and only finetunes a few of its final layers. This has proven to make models with better performance than the ones that are randomly initialized.

In this study, we use the AlexNet and ResNet [27] networks for the choice of our backbone feature extractors trained by the SiamRPN++ tracker [8].

2.2 Traditional Computer Vision Algorithms for Feature Extraction

Before the rise of deep learning era, traditional trackers had to track features extracted from images with other means than neural networks. Those operations were primarily based on mathematical operations with different hypotheses in mind. According to the success rates and F1 scores submitted for different trackers in datasets such as VOT 2021 [57], OTB [9], UAV [53], etc. trackers that use neural networks outperform the ones only relying on low level image features. This is due to the fact that neural networks many non-linear operations on every image and are better capable of extracting semantic features in them. It is also worth noting that the methods used by traditional trackers, such as filter optimization, correlation operation, key point identification, region proposal algorithms, etc. are still used by the deep learning-based trackers. Here we review a few traditional feature extraction methods used in literature.

2.2.1 Edge Detection

In traditional computer vision techniques, object edges were essential features for many reasons. They show different object boundaries and are helpful for tasks such as segmentation, detection, tracking, etc. One of the ways that object edges can be detected is through the differentiation of every pixel with its neighbors. Inside object contexts, pixels have relatively the same value. For instance, the sky color does not usually have abrupt changes. It is coherently blue or cars' color values remain relatively the same along with different parts of the car. As a result, if we subtract a pixel value corresponding to a point in the middle of the sky or a car from its adjacent pixels, the resulting number will be very close to zero.

2.2.2 Laplacian Kernel for Edge Detection

On the other hand, if we subtract a pixel color value on the edge of a car from its neighbor pixels, because the background typically has a different value, it will result in a high number. The Laplacian operator for edge detection performs exactly the same. For performing the differentiation mentioned above, kernels correlate with the image that we want to find edges in. Figure 2.12 shows two such kernels for the Laplacian operation. Typically, the Laplacian kernel is multiplied by gaussian coefficients to prevent identifying too many false edges.

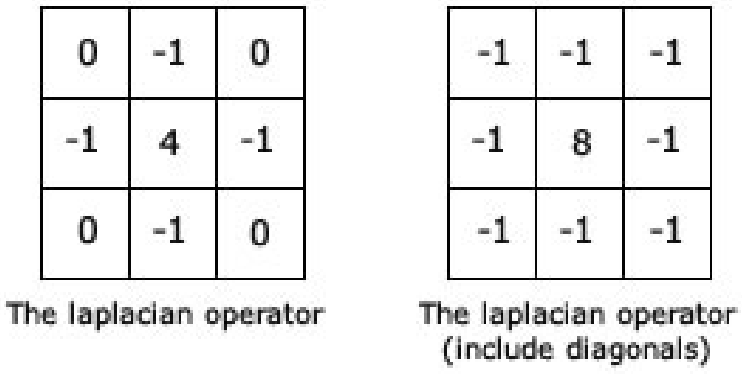

Figure 2-12: Laplacian kernels for edge detection.[28]

2.2.3 Corner Detection

Traditional trackers had to rely on features extracted by traditional computer vision algorithms to detect the target. Unfortunately, raw edges cannot be very helpful for object tracking. On the other hand, corners are better features for describing the target, and it is based on edge detection. Technically, a corner is where two strong edges in different directions meet. Corners are better features for detecting targets as they provide more concrete information about target location than the object edges. Figure 2.13 shows corners detected by Harris corner detection [29].

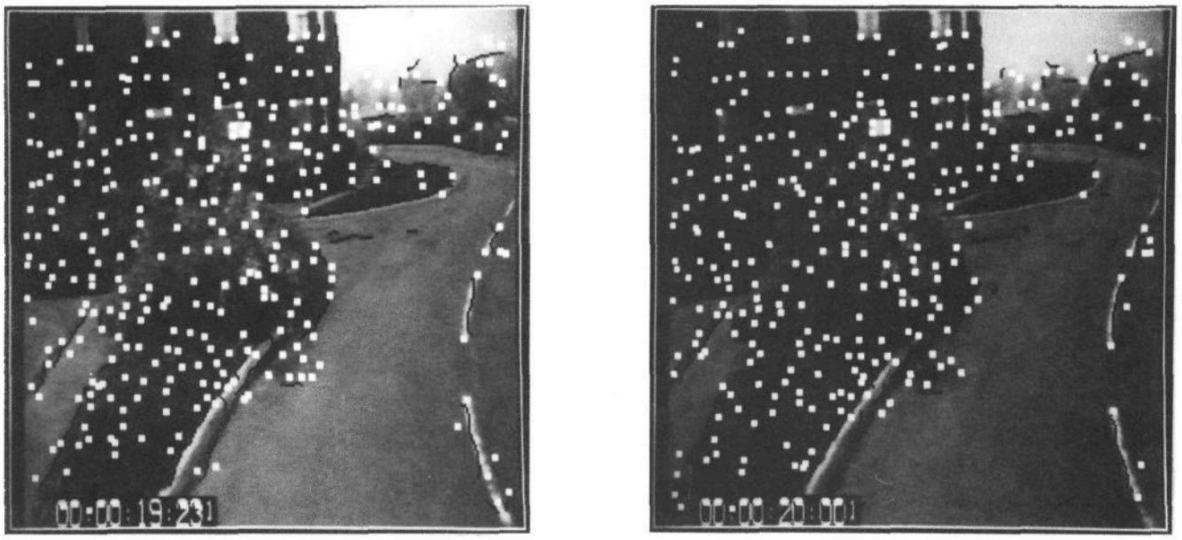

Figure 2-13: The white points illustrate the corners detected by Harris corner detection. [29]

Corners are among the most critical key points extracted from images. Many traditional trackers, such as the GOA tracker by Veenman et al. [30], were based on key point detection. Moreover, many detection algorithms such as the SIFT (Scale Invariant Feature Transform) [31], are based on corner detection methods.

2.3 Object Tracking and Detection

2.3.1 Correlation Based Trackers

Another approach that traditional trackers primarily used was the use of correlation filters. This approach has remained useful among deep learning methods for object tracking. The main difference is that deep learning methods use it to correlate the features extracted from their networks. In contrast traditional computer vision based trackers did this by correlating pixel-level information.

To use correlation filters for detection and localization, one can first normalize all the values in the target image and the filter to be between -1 and 1. Then by performing cross-

correlation, a feature map is obtained. The highest number in the feature map corresponds to the filter center in that image.

Even though correlation filters seem straight forward, performing correlation is quite time-consuming. To improve the speed of this process, many researchers take advantage of the fact that cross-correlation in the space domain is equivalent to multiplication in the Fourier domain. As a result, they first take both the target image and the filter to the Fourier domain using methods such as Fast Fourier Transform (FFT). After that the filter and image should be multiplied in the Fourier domain. Finally, the result should be transformed back to the space domain using Inverse Fast Fourier Transform (IFFT). Equation (2.6) shows the correlation in the Fourier domain

$$F(f) \odot F(h)^{*} = F(g) \tag{2.6}$$

where F denotes the transform of input to the Fourier domain, f denotes the image and h denotes the kernel (filter), * denotes the complex conjugate, and the circled dot is the element-wise multiplication.

After having the results in the Fourier domain, the output F(g) is transformed back with IFFT and produces the final score map.

Using a naive correlation filter is not robust to change of view, change of illumination, rotation and other difficulties in object tracking. However, many trackers made adaptive filters to address target changes and made fast and robust pipelines. In the next section there is an overview of some of the old trackers, especially those that used the correlation filter technique.

There is also a brief review of object detection methods, as detection is one of the most critical stages of tracking algorithms.

Before 2016 when the use of deep learning was not widespread in object tracking, most of the trackers were based on correlation filters. The main advantage of tracking with correlation filters over other traditional methods is its high speed and kernel's ability to adapt to the changes in target appearance. MOSSE tracker [1] was among the first successful works that took advantage of these two techniques. The authors proposed an approach to produce a filter using a few samples in that work. According to equation (2.6), the score map is obtained by multiplying the filter and image in the Fourier transform. So, if we define a desired training score map g_i, take it to the Fourier domain and perform an element-wise division as explained in equation (2.7), we can produce a desirable output.

$$H_i^* = \frac{G_i}{F_i} \tag{2.7}$$

where H^* corresponds to the complex conjugate of the template in frequency domain, F_i denotes the search image and G_i denotes the score map in frequency domain. The capital letters mean the Fourier transform of their lower-case features, and the letter i denotes the training sample. The error between the desired and actual output should be minimum for generating a filter with the best performance. In other words, the result of correlating the filter with a new frame in a video should have minimal difference from the actual score map. Equation (2.8) shows the mathematical formulation of this issue:

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{2.8}$$

where filter H^* is designed in a way to minimize the sum of squared error over the training samples. To minimize the sum of squared error, the partial derivative of this function with respect to H^* is computed and set equal to zero. Equation (2.9) is used to minimize the error. In that equation, elements of H^* are denoted by ω and ν, and the equation is solved for them individually.

$$0 = \frac{\partial}{\partial H^*_{\omega\nu}} \sum_i |F_{i\omega\nu} H^*_{\omega\nu} - G_{i\omega\nu}|^2 \tag{2.9}$$

where ω and ν are H^* elements. Equation (2.9) can be simplified, and the derivative can be solved which leads to the equation (2.10):

$$H^*_i = \frac{G_i}{F_i} = \frac{G_i \odot F^*_i}{F_i \odot F^*_i} \tag{2.10}$$

where i denotes all training samples extracted from the first frame. It is shown in practice that relying only on single templates of training samples leads to poor tracking. For solving this issue, the filters are averaged during the initialization using equation (2.11):

$$H^* = \frac{1}{N} \sum_i \frac{G_i \odot F^*_i}{F_i \odot F^*_i} \tag{2.11}$$

where N is the number of samples in the training set extracted from the frame, which is set to eight in the MOSSE tracker. Each of those eight training samples is generated by applying random affine transform to the target.

Although the tracker is well initialized, this single filter cannot track the target during different obstacles, such as changes in scale, illumination, partial occlusion, rotation, changes in pose, etc. Thus, it is vital to update the template in every frame. The MOSSE tracker computes the running average of the filter obtained in every frame with the available filter to update the filter. This is done through equation (2.12) using a learning rate of η, normally set to 0.125.

$$H^* = \frac{A_i}{B_i} \tag{2.12}$$

where A_i and B_i are defined as:

$$A_i = \eta G_i \odot F_i^* + (1-\eta) A_{i-1} \tag{2.13}$$

$$B_i = \eta F_i \odot F_i^* + (1-\eta) B_{i-1} \tag{2.14}$$

and η is the learning rate. According to that paper, this value for learning rate allows the filter to update itself quickly and result in robust tracking without letting the filter mistakenly track the background instead of the target.

Figure 2.14 shows the filter and the score map made by the MOSSE and some other filters in a video:

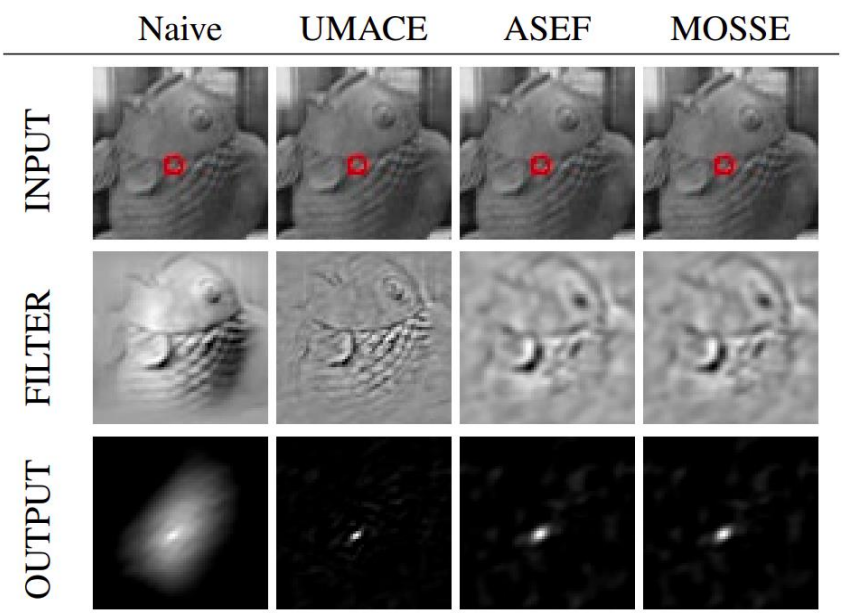

Figure 2-14: MOSSE and other filters in practice. [1]

For detecting failure in tracking, a simple method named PSR (Peak to Sidelobe Ratio) is used. PSR shows how much the peak value, which is the target position, is higher than the sidelobe. It is worth noting that the sidelobe excludes the 11x11 window adjacent to the peak. The PSR value is computed using equation (2.15):

$$\frac{g_{max}-\mu_{sl}}{\sigma_{sl}} \tag{2.15}$$

where g_{max} is the maximum value of the score map g, μ_{sl} is the mean of the sidelobe and σ_{sl} is its standard deviation. According to Bolme et al. [1], a PSR value between 20 and 60 corresponds to robust tracking, and around seven PSR values is considered as either tracking

failure or occlusion. The python implementation of this tracker can run at speeds close to 670 FPS depending on the bounding box size.

After Bolme et al. [1] proposed the MOSSE tracker, many others improved their work. The KCF tracker proposed by Henriques et al. [32] generalized this idea over all image channels. Histogram of oriented gradients (HOG) features are also used to improve the tracker results further. According to their paper, the KCF tracker runs at 172 frames per second. It is prone to losing the target in case target appearance changes quickly when started to track. CSR-DCF [33] is another successful tracker based on correlation filters. They introduced the concept of channel and spatial reliability, leading to robust tracking at speeds close to real-time. Some algorithms, such as ACI [34], combine different trackers without relying on first frame information to improve the tracking results. However, since these methods operate on low level features they are not able to recognize and use deep semantic features of targets. To address this issue, deep neural networks help by extracting those features using a trained network.

2.3.2 Deep Learning Based

One of the most famous two-stage detectors is the Faster R-CNN [35]. Firstly, features are extracted using a backbone network and fed into a region proposal network (RPN) to find which parts of the feature map might contain an object. Then the selected feature maps are embedded and fed into a classifier network to predict which class the input belongs to. Even though it is not as fast as one-stage detectors such as YOLO [37], its architecture resulted in accurate detection and motivated many others to build upon their work.

EfficientDet [36] is another object detection algorithm that is designed with the purpose of working with low inference time and obtaining accurate results. Similar to YOLO [37], EfficientDet is a single-stage detector. They used EfficientNet [38] backbone and proposed Bi-directional Feature Pyramid Network (BiFPN) and a compound scaling method to optimize the

network and achieve state-of-the-art results in object detection. According to practical observations, when trained on a relatively low number of images, its performance is worse than other convolutional based detectors. Having fewer parameters also causes networks to lose their ability to generalize on unforeseen datasets.

The use of Siamese networks is a common approach for object tracking. Before Siamese networks became popular in object tracking, they had been used for other tasks. Bromley et al. [6] proposed the first Siamese network for signature verification. It was made of two identical networks joined at their outputs. Since then, many have used this network architecture for different purposes. Schroff et al. [7] used this architecture for face recognition. They also introduced triplet loss to better train Siamese networks. Bertinetto et al. [39] were the first to use Siamese network for object tracking and made the SiamFC tracker. In that work, they first extract image features using a fully convolutional Siamese network and localize the target by cross-correlating the feature maps. Figure 2.15 shows the architecture of the SiamFC tracker [39]. This tracker is not able to properly handle changes in size as it only performs cross-correlation between outputs of the networks, unless the same image with different scales are fed into it.

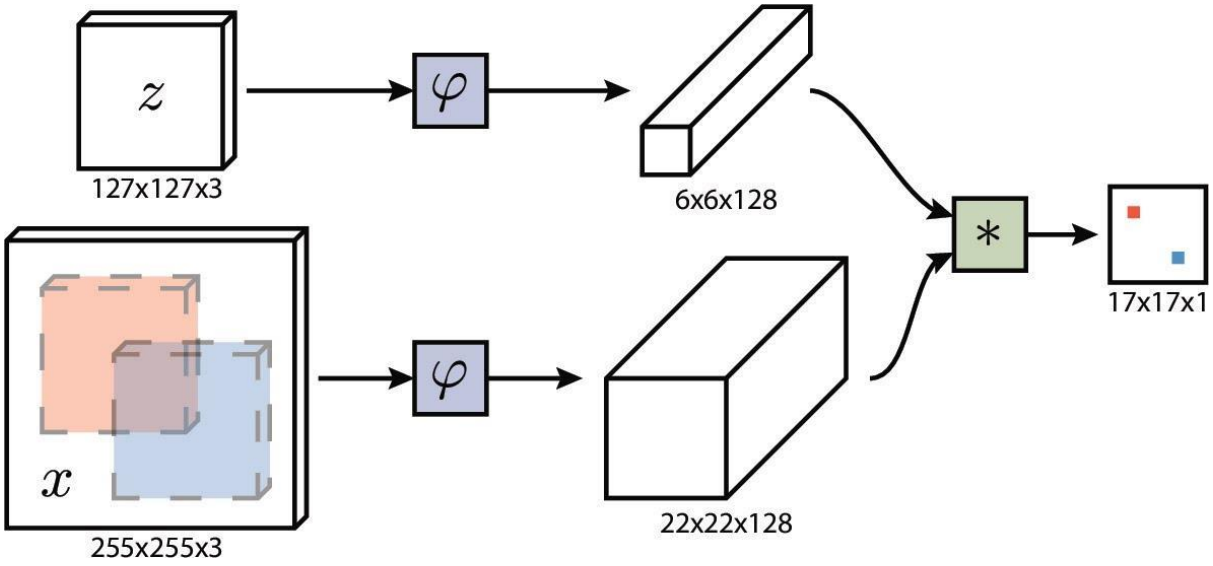

Figure 2-15: Overall architecture of SiamFC [39]. The two convolutional networks are completely identical and their outputs are combined using a cross-correlation. The highest value in the output feature map shows target location in the search image. Image reprinted from [39].

SiamRPN tracker [16] adds a region proposal network to a Siamese backbone to perform correlation only between the proposed anchors and the target template. SiamRPN++ tracker [8] improves the SiamRPN [16] tracker and uses deeper networks for feature extraction. Despite all the improvements, it is unfortunate that they use only target features in the first frame as their template images. Motivated by Bolme et al. [1] that updates the target template in every frame where the target is detected, in this work we replace the templates with newer ones to capture both recent and diverse appearances of the target.

2.3.3 Short-term and Long-term Tracking

Short-term tracking and long-term tracking are different by nature. While short-term tracking requires accurate model updating for addressing the continuous changes in target appearance, long-term tracking mostly needs the correct identification of the target. For this reason, most short-term trackers focus on designing models that can handle target changes with

high inference speeds, while long-term trackers focus on precisely identifying the target. Wrong updates in long-term tracking can be quite dangerous as it will cause the model to lose the target in long videos. Here we review state-of-the-art short-term and long-term trackers.

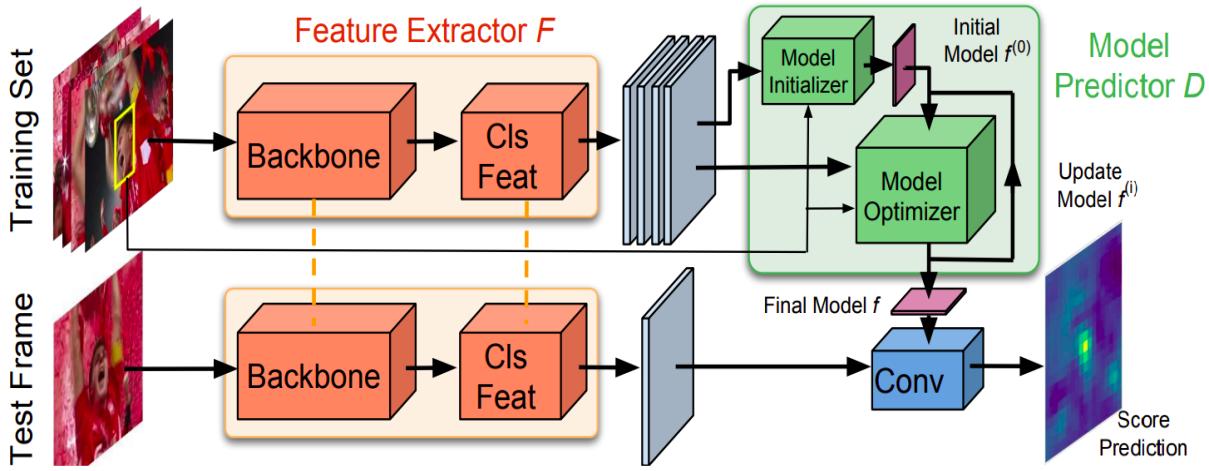

Figure 2-16: An overview of the DiMP tracker [2]. As seen in the image, unlike many other trackers, the training set for this tracker contains both the target and the area around it, which enables it to use background information to distinguish the target from the background. Image reprinted from [2].

Goutam Bhat et al. [2] proposed the DiMP tracker for short-term object tracking, outperforming most trackers. In that work, they extract features of the training set, a set of target images whenever the confidence is above a certain threshold. After feature extraction from the training images, a model is predicted and optimized using their designed model optimizer. Then they extract the features from the test frame and perform a convolution operation between the predicted model and the extracted features. Using a ResNet 18 as a backbone, this method can run at 57 frames per second on a single Nvidia GTX 1080 GPU. The main reason for their high speed, robustness and accuracy is the template updating which does not stop the model from using later extracted features. Figure 2-15 shows the tracker

architecture in details. A problem that this tracker suffers from is wrong updates when it produces false positive. Wrong update leads to a wrong optimized model which loses the target and tracks the background instead.

Siam R-CNN [3] is an example of a long-term tracker. In that work, first they extract test image and ground truth features, i.e., features of the target in the first frame, using a backbone with the shared number of layers, weights, etc. Then ground truth features are aligned using the RoI align module first proposed by the Mask R-CNN [4] detection algorithm. After aligning the features, they feed them to the redetection head to perform classification based on their similarity and regression for accurate bounding box coordinate extraction. After doing the same operations with the template extracted from the target in the previous frame, they use the Tracklet Dynamic Programming Algorithm. That algorithm tracks both the target and all possible distractors within tracklets. Each tracklet tracks an object for several frames before the object disappears. The target is segmented in the test image using a network named Box2Seg (bounding box to segmentation), which was first introduced by J. Luiten et al [5].

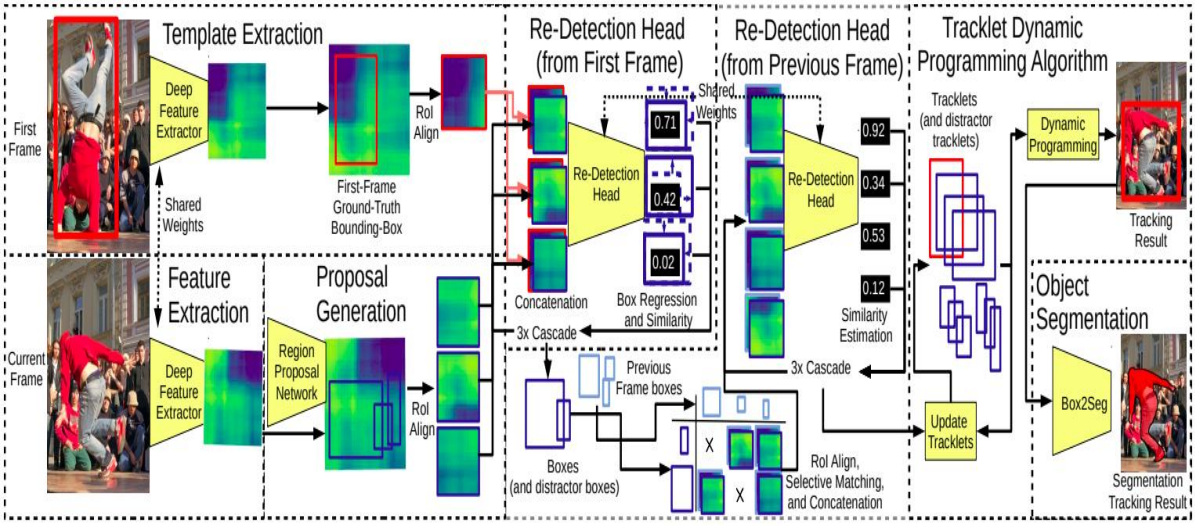

Figure 2-17: An overview of how the Siam R-CNN [3] tracker works. It uses the target template in the previous and the first frame to see if the proposed regions of interests (ROIs) contain the target or not. Image reprinted from [3].

A problem that Siam R-CNN suffers from is producing false positives when the target is not among the objects the region proposal network (RPN) was trained to identify. In some cases it continuously switches between the true target and distractors no matter how far they are from each other. Even though Siam R-CNN is a long-term tracker, similar to other trackers of this type, it achieves relatively good results on both short-term and long-term tracking benchmarks. However, most short-term trackers cannot achieve acceptable results on long-term datasets. The main reason for this issue is that they do not have a detection phase as strong as the one used in long-term trackers, and they might not understand the absence of the target well enough. Updating the template while the target is not present is a big problem most short-term trackers suffer when tested on long-term benchmarks.

A successful short-term Siamese-based tracker is SiamRPN [16] tracker. It is consisted of two major parts: a Siamese backbone and a region proposal head. Figure 2.17 shows the architecture of the SiamRPN tracker [16]. Firstly, the Siamese backbone extracts features from

both the detection frame and template frame in that network. Then the features are fed to the region proposal subnetwork. The RPN module changes the size of search and template frames. It also adds k anchors to the shape of the features extracted from the template frame. By performing cross-correlation, that network produces k groups of feature maps in classification and regression branches. Each of those feature maps correspond to one of those k anchors. Every anchor produces two feature maps for the classification branch, indicating whether the specified location contains the target or the background. The number of feature maps generated by every anchor in the regression branch is four, indicating the four corners of the fine-tuned bounding boxes. However, in that tracker using deep networks does not improve the results due to problems in training.

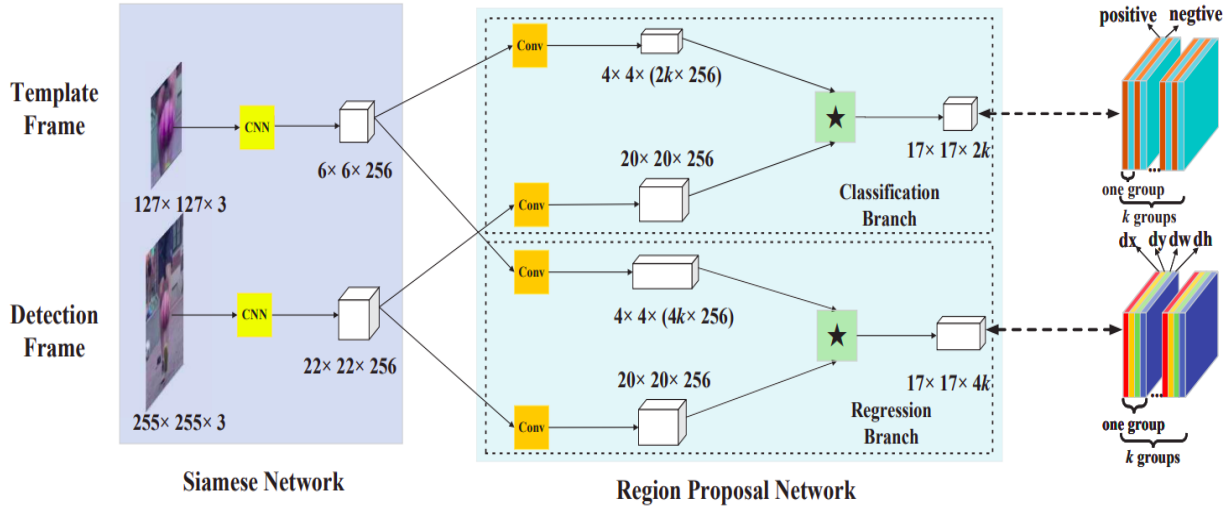

Figure 2-18: Overview of the SiamRPN architecture. After Siamese backbone networks extract features from both template and search image, the region proposal network produces classification and regression feature maps. The image is taken from Li et al. [16].

SiamRPN++ [8] tracker is a long-term tracker designed based on the SiamRPN [16] with three major differences: the use of bigger backbones, performing depth-wise cross

correlation instead of up-sampling convolution and the use of multiple heads using the features of different layers of backbones. Two of the most important backbones used in SiamRPN++ [8] are AlexNet [19] and ResNet 50 [27]. AlexNet is a small convolutional neural network which makes possible tracking with high speed. It was the first convolutional neural network with the use of ReLU and dropout to achieve outstanding results on ImageNet 2012 [52] classification. Its architecture design also allows to use 2 GPUs simultaneously. Residual networks [27] were first introduced in 2016 and have greatly enhanced the field of computer vision, as they made possible the use of very deep networks with over 1000 layers. Figure 2.18 shows the AlexNet [19] architecture and Figure 2.19 shows the building block of ResNet 50 [27].

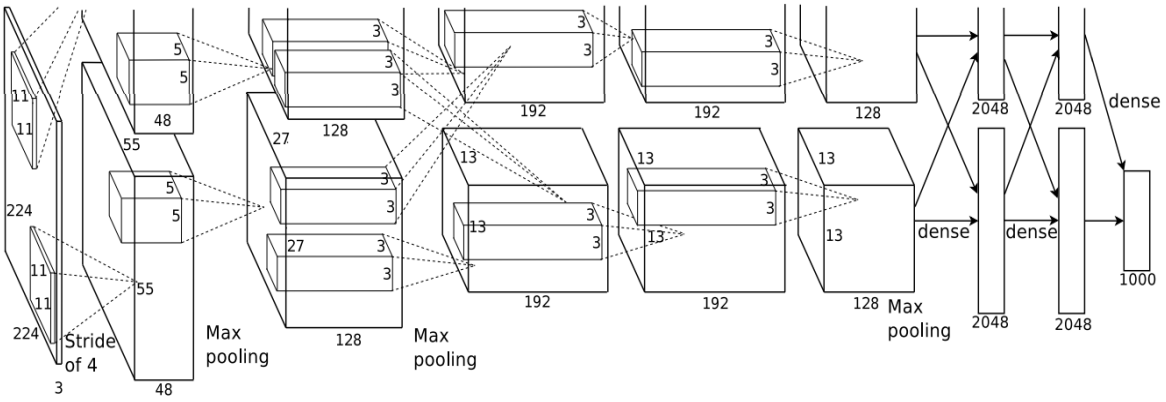

Figure 2-19: AlexNet architecture. The input size is 224x224x3. Due to its architecture, it can use two GPUs with different responsibilities simultaneously. One GPU processes the top layer parts of inputs and the other processes the bottom parts. In the end, the two feature maps are concatenated using dense layers. Image reprinted from [19].

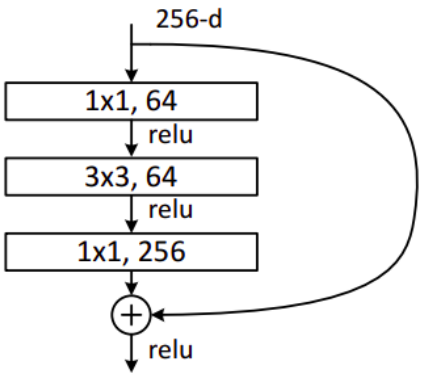

Figure 2-20: Building block bottleneck of ResNet 50 architecture. The residual building blocks allow the training of very deep networks with more than 1000 layers without facing issues such as gradient exploding or vanishing: Building block bottleneck of ResNet 50. Even though a very high number of layers can be used, according to He et al. [27] its performance is not as good as using the architecture with 101 layers. Source: [27]

To better illustrate the differences between SiamRPN++ [8] and SiamRPN [16] tracker, Figure 2.20 is provided. It shows the difference between the types of cross-correlation used in those two trackers. While SiamRPN uses up-channel cross-correlation that is a time-consuming process, SiamRPN++ uses the depth-wise cross-correlation. Depth-wise cross-correlation uses 10 times less parameters than the up-channel cross-correlation and has the same performance.

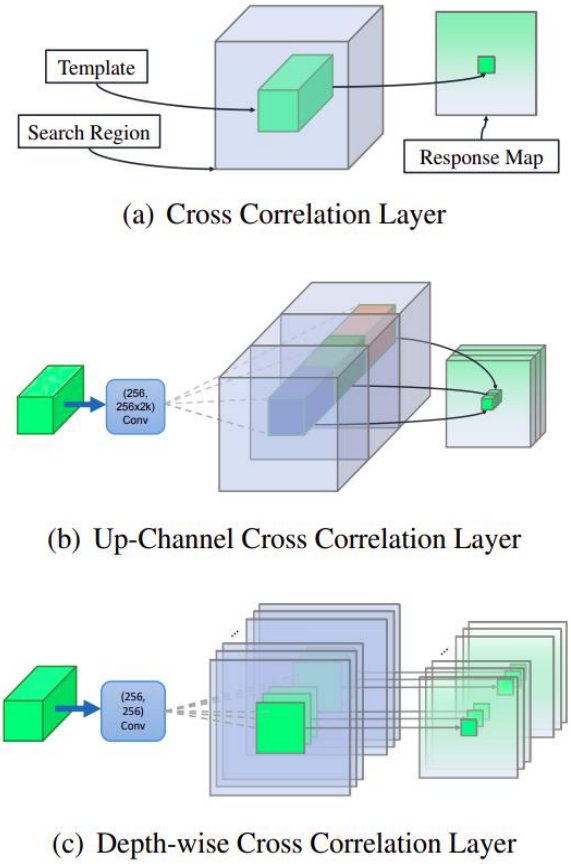

Figure 2-21: Three different types of cross-correlation used by different trackers. a: conventional cross-correlation used by trackers such as SiamFC [39]. b: Up-channel cross-correlation used by trackers such as SiamRPN [16]. c: Depth-wise cross-correlation used by trackers such as SiamRPN++ [8]. Image reprinted from Li et al. [8].

Despite SiamRPN++ [8] superiority over SiamRPN [16] and many other detection-based trackers, it does not use the recent target appearances and it causes the true strength of this tracker remain unused. Next chapter includes our methodology and includes how we solved this issue using our proposed bag of dynamic templates.

Chapter 3. Methodology

This chapter provides in-depth information about our proposed tracker Multi-Template Temporal Siamese network with candidate selection algorithm. This work is based on the SiamRPN++ tracker [8] with modifications that improve the results. The proposed candidate selection algorithm also improves detection-based trackers results.

In this study, there are three main contributions, and each section of this chapter belongs to one of them.

Section 3.1 provides an overview of the Multi-template Siamese network for measuring similarity and localizing targets. It also provides an overview of the backbones used and their

Siamese architecture. After that the region proposal network and how it uses features to localize the target in images is discussed. Finally, there is a discussion over how templates update and a proper set of hyperparameters for different backbones for replacing the templates with the most recent and diverse information.

Section 3.2 presents the temporal network trained for the task of bounding box regression, its architecture, the loss function used for that task, and the techniques used for training that network.

Section 3.3 provides in-depth information about the candidate selection algorithm, its mathematical formulas, and why we generated the formulas in this work. Finally, a brief discussion of the effect of each of the hyperparameters used in the candidate selection algorithm.

3.1 Multi-Template Siamese Network

Firstly, similar to the SiamRPN tracker [16], a Siamese network is designed to extract features from each search image and the template image. In the SiamRPN tracker, the feature extractor is a modified version of the AlexNet [19] convolutional neural network. Due to the simplicity of AlexNet and its few numbers of layers, it results in tracking with high speed; however, since the features extracted by this network are not as good as ResNet50 [27] or VGG16 [40] networks, the region proposal network built on top of it does not perform as well.

The Multi-Template Siamese part of our work is similar to the SiamRPN [16] network with three differences:

Firstly, in the original SiamRPN tracker, there is only one template image of the target, but we use *n* different template images containing the target in different frames. This allows us

to track targets when different challenges such as fast target movement and occlusion are happening.

Secondly, the SiamRPN tracker only uses modified AlexNet as the backbone with a single RPN head. When we use the ResNet 50 as our backbone there are three RPN heads. The input to each of those heads is the output of different layers in the Residual backbone network. Interestingly, according to Li et al. [8], who proposed the SiamRPN++ tracker, different objects have high responses in the same channels if they are from the same class, making it essential to have the RPN head at different times layers. The final feature maps are computed by taking a weighted average of those three RPN modules.

Thirdly, in the SiamRPN tracker [16], convolutional networks are scaling up input channels for performing up-channel cross-correlation. Instead of using up-channel cross-correlation, feature adjustment and depth-wise correlation are used, substantially reducing the computation cost while maintaining the same scores. Figure 3.2 illustrates the differences between depth-wise cross-correlation, up-channel cross-correlation, and conventional cross-correlation. The depth-wise cross-correlation is used by the SiamRPN++ tracker [8] as well.

In conventional cross-correlation, the number of template and search region feature channels is the same. In the up-channel cross-correlation, the number of channels in search image features is a fraction of the feature channels of the template image in the SiamRPN [16] tracker. In this work and the SiamRPN++ tracker, the features should be adjusted to have the same number of channels before performing depth-wise cross-correlation. Then the conventional cross-correlation operation is performed channel by channel. The resulting features are then fused using a convolutional layer.

Figure 3.1 shows an overview of our work and its different sections. Firstly, unlike SiamRPN++ [8] which uses a single target image as the template, we use a bag of templates

captured from the target in different frames. The bag replaces target old appearances with the new and diverse ones. The images are fed into the Siamese backbone and the RPN head provides target candidates in the search image considering all images in the bag of templates. It provides a classification feature map and a regression feature map. The classification head's feature map shows the probability of whether the proposed bounding box contains the target or the background. In this work, we compute the average of all classification feature maps to combine the results of all templates in the bag. A temporal network is trained to learn the bounding box coordinates in different frames. It makes a predicts where the target might be given its coordinates in the previous four frames. Finally, the candidate selection algorithm decides which candidate is the most reliable by having classification scores, bounding box coordinates, temporal network prediction, and internal parameters. The formulas used to design the candidate selection process are explained in section 3.3. After target detection in a new frame, the bag of templates must get updated. The target score must be above a certain threshold for updating the bag of templates. For this purpose, the target reliability score is not used. Instead, the classification score is used as it shows how similar its Siamese features are to the templates in the bag. To localize targets by the Siamese network, we avoided training new models. Instead, we used the trained networks by Li et al. [8]. The loss function used by them is the sum of smooth L1 loss and cross-entropy loss.

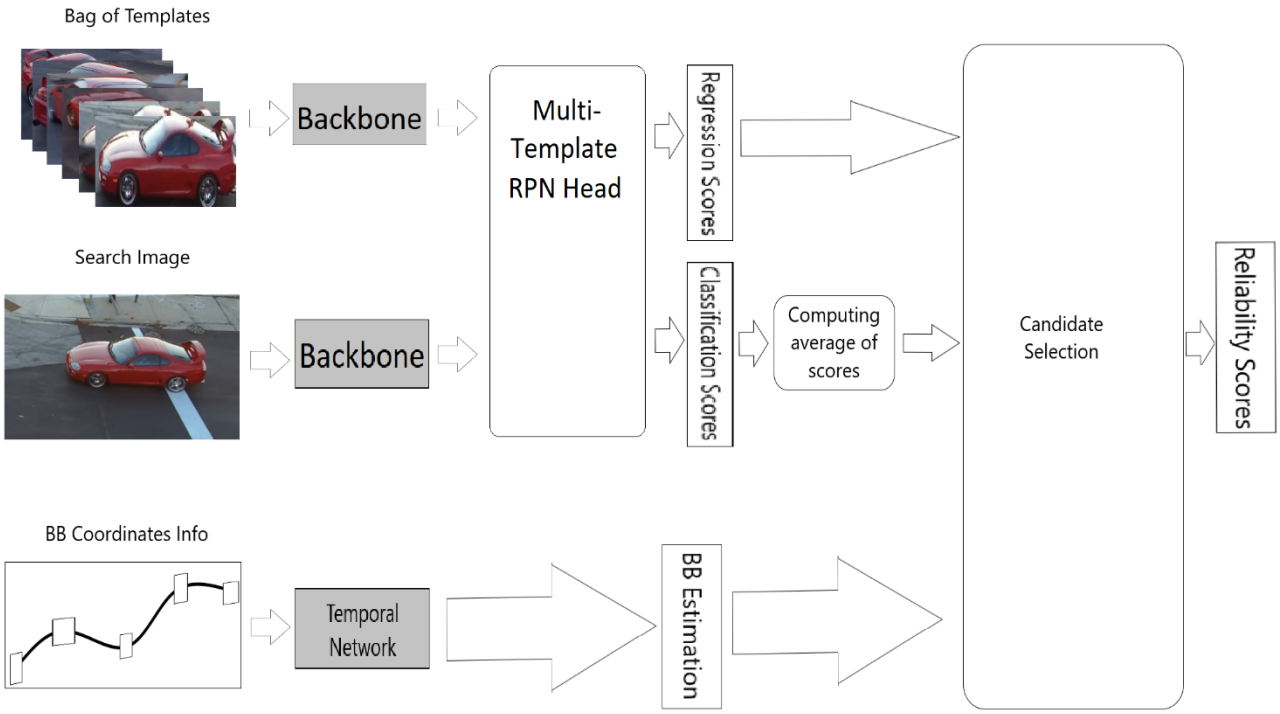

Figure 3-1: An overview of the Multi-template Siamese network with Candidate selection algorithm. Given a bag of target templates and the search image, the RPN heads propose candidates. The candidate selection algorithm finds the most reliable one using predicted bounding box information.

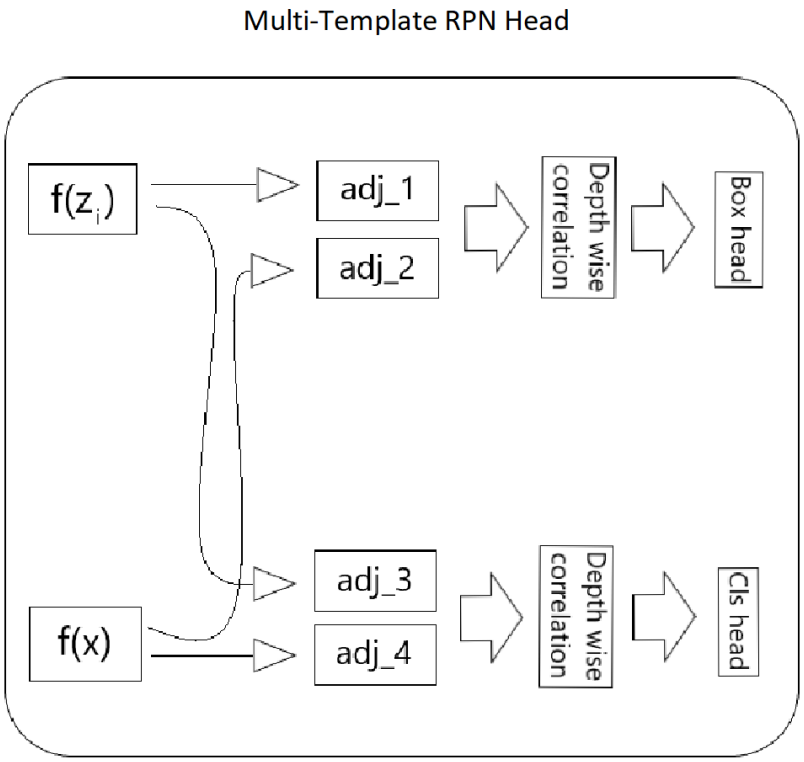

Figure 3-2: An overview of the Multi-Template RPN Head. f(zi) is the ith template features extracted by the backbone, f(x) is search image features and adj corresponds to feature adjustment before depth-wise cross-correlation.

In this work, two of the chosen backbones are AlexNet [19] and ResNet 50 [27]. While AlexNet [19] is chosen for its high speed, ResNet [27] is preferred for its better results. AlexNet convolutional network architecture was designed in 2012 for ImageNet [52] classification whereas ResNet architecture was proposed in 2016 and is relatively newer than AlexNet. ResNet also has more layers than the AlexNet, allowing it to extract more features from inputs. It is important to use features extracted by good backbones for updating the bag of templates. Another option is the use of a MobileNetV2 [58] where there is a trade off between speed and accuracy. This network is designed to have a good performance with low computations for devices such as cell phones, etc. However, since it is not as fast as AlexNet [19] or accurate as ResNet50 [27] we avoid using this network in this work.

For updating the bag of templates, we followed two strategies: Replacing templates with the most recent and similar ones and replacing templates with the carefully selected diverse ones. In short-term tracking, it is essential to address target changes within a short time. It also does not require the tracker to have a highly sophisticated re-identification module. So, we used the first strategy for short-term tracking datasets. We used the second strategy for long-term tracking as the target might reappear with a shape different from how it was last seen. We selected target templates in frames whose classification score was slightly over the threshold to find diverse and reliable target features. In other words, updating templates depends on the current classification score. The first template always contains the target appearance in the first frame and never gets updated. The second template always contains target features when the classification score is very close to 1, i.e., above 0.9999. The third template updates when classification score is below the second template threshold and above a low threshold, resulting the template 3 to update on normal conditions when classification score is neither high or low. In this way, the third template contains recent target features. The update of fourth and fifth templates depend on tracking type. When performing short-term tracking, they also contain common numbers for classification score When performing long-term tracking, the fourth, fifth and sixth template update when classification score is above minimum threshold to contain diverse target features. For instance, fourth, fifth and sixth templates can update when classification score is above 0.91, 0.88, 0.85 without overlaps respectively. In this way, diverse features of targets are stored.

For the total number of templates, we tried six and ten templates. According to our experiments, increasing the number of templates from six to ten slightly improved the results but also reduced the speed. Figure 3.2 shows how the bag of templates updates during long-term tracking.

Figure 3-3: An overview of how the bag of templates updates during long-term tracking. Template number 1 never changes as it contains ground truth information. Other templates are responsible for containing various target features.

3.2 Temporal Network

One of the problems that we observed most of the detection-based trackers, mainly Siamese based trackers have, is that they continuously switch between the target and possible distractors. For instance, Figure 3.3 shows how the SiamFC tracker [39] mistakenly tracks other similar objects to the target. SiamFC is one of the most famous Siamese based trackers. It uses only target features in the first frame to detect it in the current search frame. Since it does not update its template and ignores target path information, it finds many different objects similar to the target and tracks them instead of the actual target.

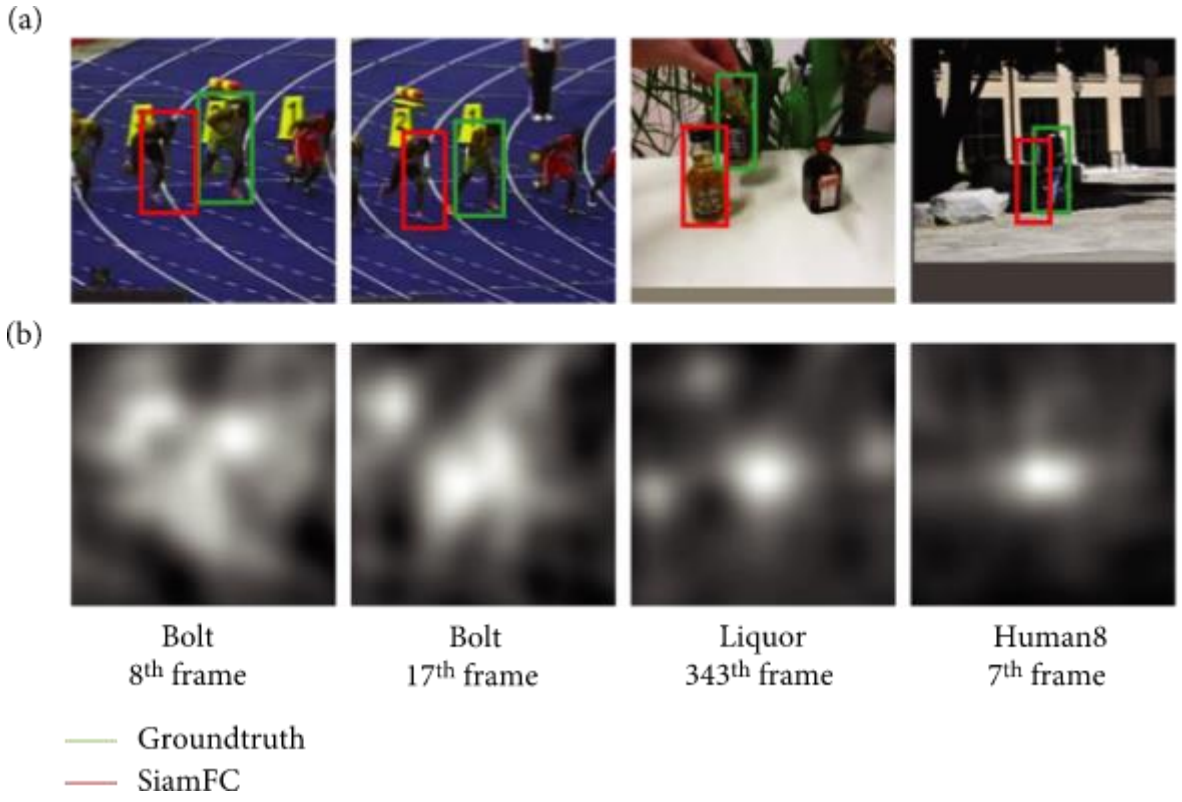

Figure 3-4: Some examples of SiamFC failure due to the existence of distractors in the search image. (a) shows the search frame and (b) shows the score map produced by that tracker. [42]

To solve the issue mentioned above, many approaches can be tackled. This study proposes one of the most straightforward and computationally most efficient approaches using target bounding box prediction. We believe that by estimating over the target position in a

search frame, the target can be better selected among possible candidates. To this end, the first step is to train a network that can predict the target position using its bounding box coordination in previous frames. We performed extensive experiments on the GOT10K dataset and trained many different networks with different hyperparameters. We also evaluated input size and observed how it changes network results. We realized that a six layer network with bounding box coordination of the previous *f* frames produces the lowest error on the training and evaluation set of the GOT10K dataset. In our experiments, we set *f* to 4.

Temporal neural networks are a type of neural networks in which the input varies over time. Lea et al. [43] proposed the first temporal convolutional network for the task of action segmentation in videos. In that work an encoder-decoder convolutional network processed the input across time, leading to satisfactory results.

The temporal network in this study uses bounding box center coordinates and its width and height in every *f* sequential frames and predicts the coordinates in the next frame. At least four values are needed to produce a bounding box in a two-dimensional image. The 4 values which this network predicts are the top left *x* and *y* coordinates of the bounding box, as well as its height and width. Since this network predicts the 4 required values, it can also predict target scale change as well. The loss function used for the temporal network is the L1 loss (equation 2.3), where the sum of absolute error between prediction and ground truth is computed and used for backpropagation.

To train the network and perform the regression task, we observed that images in the GOT10K dataset have different sizes. This causes a huge imbalance in the training dataset, leading to poor perception of changes in bounding box appearance and coordinates through time. For instance, in a 256x256 image, the coordinates for the middle of the image are (128, 128), whereas these coordinates for a 1920x1080 refers to the top left part of the image. Also,

the target moving speed computed by pixels per frame is different when the image resolution is different. Suppose a target within a certain distance from a camera moves a step in a direction. If we use low-resolution images, the target has moved only a few pixels, as each pixel captures relatively a big space. Using high-resolution images shows that the target has moved so many pixels towards that direction since each pixel covers a relatively small area. To solve these issues, we normalized the coordinates in each image with respect to its resolution. This makes the regressor predictions be values between 0 and 1. It also makes the changes in the bounding box coherent when using pixel coordinates as our metric. According to our experiments with kalmann filter for target coordinates prediction, when the target was absent the predictions were rather linear which was one of the reasons that encouraged us to make this network.

3.3 Candidate Selection Algorithm

After obtaining approximate target position and some candidates with different bounding boxes and classification scores, we should select the most reliable candidate. Not only does the candidate selection algorithm selects the most reliable candidate, but it also prevents the tracker from mistakenly switching between the target and possible distractors. Among the candidates proposed by the region proposal network head, we select the top 10 ones with the highest classification scores. After that, we need to measure how far they are from the predicted coordinates. We call this difference the distance error. Equation (3.1) shows how distance error is computed:

$$DE_j = \frac{\left|\sum_{i=1}^{4} PC_i - \sum_{i=1}^{4} CC_{j,i}\right|}{2} \tag{3.1}$$

where DE_j is the distance error of each one of those ten candidates. PC_1,PC_2, PC_3 and PC_4 are the top left corner x and y, height and width of the predicted coordinates, respectively. $CC_{j,1}$, , $CC_{j,}2$, $CC_{j,3}$ and $CC_{j,4}$ are the top left corner x and y, height and width of predicted coordinates

of the *j*th candidate coordinates. The Distance Error value is always between zero and one to simplify further computations.

After computing the distance error for each candidate, the reliability score can be computed. The reliability score considers each candidate's classification score, its distance error, a sequence confidence parameter and a weight that determines how much effect the temporal path prediction network should have over the classification score. Equation (3.2) shows the mathematical formula for computing each candidate's reliability score:

$$RS_j = C_j - \frac{(DE_j - T) * (SC_t) * (1 - C_j)}{RW} \tag{3.2}$$

where RS_j, is the final reliability score for the *j*th candidate. C_j is the classification score for candidate *j* which shows its similarity to the target. Robust Weight, RW, is the weight specified for tuning the effect of temporal network prediction on the final reliability score. T is a threshold to increase the reliability score when predicted coordinates are quite close to a candidate coordinates. In our experiments, we set T to 0.1. SC_t denotes the tracker sequential confidence value and * denotes multiplication. Low values for RW cause the candidate selection algorithm to rely on it, whereas low high values decrease its effect. The sequential confidence value, SC, is a parameter in the candidate selection algorithm which updates itself after each frame is processed. The updating formula is equation (3.3):

$$SC_t = \frac{4SC_{t-1} + C_t}{5} \tag{3.3}$$

Sequential Confidence determines how confident the tracker has been in previous frames. It computes a moving average over different classification scores. Its default value at the

beginning of the tracking is 1. If the target classification score has been high in recent previous frames, the sequential confidence value increases; otherwise, it decreases.

According to equation (3.2), each candidate's classification score is subtracted by a value. If the distance error is high and the tracker classification score has been high in previous frames (i.e., high sequence confidence), the candidate classification score decreases by a high value. If distance error is high, meaning that the candidate is far from where it is expected to be, but the sequence confidence is low, meaning that the tracker has not been confident in previous frames, not much emphasis is put on path prediction as the candidate classification score decreases by a low number. If the error distance is low and the tracker has been confident in previous frames, the reliability score is subtracted by a low value, or its value might increase. Suppose the distance error is low but the tracker has not been confident in its previous frames. In that case, the classification scores decrease by a low number as path information cannot be much helpful in this situation. Finally, the effect of path information is multiplied by a coefficient (1 - classification) score for candidate *j*. This is because the classification score for smaller backbones such as AlexNet is much lower than the classification score of the bigger ones, like ResNet 50. This final coefficient fine-tunes the effect of the candidate selection algorithm with respect to the backbone errors. If a backbone that produces better classification scores is used, the candidate's classification score should not decrease by a large number to produce the reliability score. On the other hand, if a backbone that produces weak classification scores is used, the effect of path information should increase.

Chapter 4. Experiments

This chapter provides experiments we performed to design the tracker, select its hyperparameters and different strategies we examined for updating the bag of templates that led to the current strategy. We also provide in-depth information about how this tracker performs on different datasets and its performance on real-life videos taken by phones and uploaded online.

Section 4.1 provides technical details of our work. The details include the choice of the initial learning rate for training the temporal network, its decaying algorithm, the effect of batch size in mini-batch gradient descent and how it produced better results over stochastic gradient descent, etc.

Section 4.2 analyses model performance on different datasets. It includes numerical results and comparison with other trackers and qualitative results showing when this model fails. It also includes an ablation study on how different parts of the tracker perform individually and how their combination leads to better performance. It contains the tracker performance in a real-world video taken from a cell-phone to see how it tracks targets in practice. We compare the results on these videos with the baseline of this work, which is the SiamRPN++ tracker [8]. Finally, section 4.3 has a discussion over the speed of trackers and our MTTSaim in particular.

4.1 Technical Details

This section is made of 3 parts discussing the technical details of the three major parts of this tracker:

1. The multi-template Siamese network and the bag of features.
2. The temporal network.
3. The candidate selection algorithm.

4.1.1 Multi-template Siamese Network and the Bag of Templates

For the backbone choice, we used the pre-trained backbones of the SiamRPN++ [8]. They used a pretrained network on ImageNet [52] and trained the network on COCO [56], ImageNet DET [37], ImageNet VID [52], and YouTube-Bounding Boxes Dataset [44]. For short-term tracking, the bag of templates contains most recent and similar to diverse target appearances with a focus on recent similar ones. For long-term tracking there is more focus on the target diverse appearances to better be able to reidentify the target after long disappearances.

4.1.2 Temporal Network

We used the bounding box labels to train the network on the GOT10K dataset and did not train on the images. To train this network, we tried several different architectures. Here we review those architectures and compare their results to the GOT10K dataset.

4.1.2.1 Temporal Multi-Layer Perceptron

According to our experiments, best results are achieved when we use 6 layers. The bounding box coordinates of different frames are concatenated together to make a one-dimensional vector to train the network. That vector is then fed to the network to make predictions. We put our experiments' results with multi-layer perceptron networks in Table one to let the readers better compare our models.

MLP #	# of layers	Input size	L1 Error Validation Set
1	3	64	0.014
2	5	64	0.013
3	6	64	0.011

Table 4-1: Temporal MLP trained on ground truth information of GOT10K to predict bounding box coordinates by learning target movements. Input size is 4 times bigger than the number of frames in which their bounding boxes are used, as each bounding box in each frame can be constructed with 4 values.

We also realized that using dropout or L2 regularization improves the performance of the validation set when they are used separately. However, when they both are used simultaneously, the performance decreases.

The networks were trained for 100 epochs. We used mini-batch gradient descent with momentum set to 0.875. We also used exponentially learning rate decay and L2 regularization. In our experiments, L2 regularization technique performed better than dropout and L1 regularization for this task. We also observed that by training networks for this task with small batch sizes, the network does not converge and it should be above 400. A batch size of 32768 samples was used for our experiments.

At first, we tried the Root Mean Square Error (RMSE) loss function to train networks. But we realized that as the learning rate decreases, its error increases. According to our observations, the gradient was too big, and sometimes it was not a real number, and disrupted network parameters. We realized that as the error gets closer to zero, the derivative of the RMSE loss function increases drastically. Mathematically, if we want to take the derivative of a function around point x, $x \pm epsilon$ should be available where epsilon is a value close to zero. Because the root square of negative values does not produce real numbers, the derivative of RMSE loss when the loss is zero is not defined. When we used a logarithmic loss function, the gradient increased to infinity as the slope of the logarithmic function close to zero is close to infinity. All these problems were solved using the L1 loss, the mean of the sum of errors. Figure 4.2 shows the logarithmic function plot.

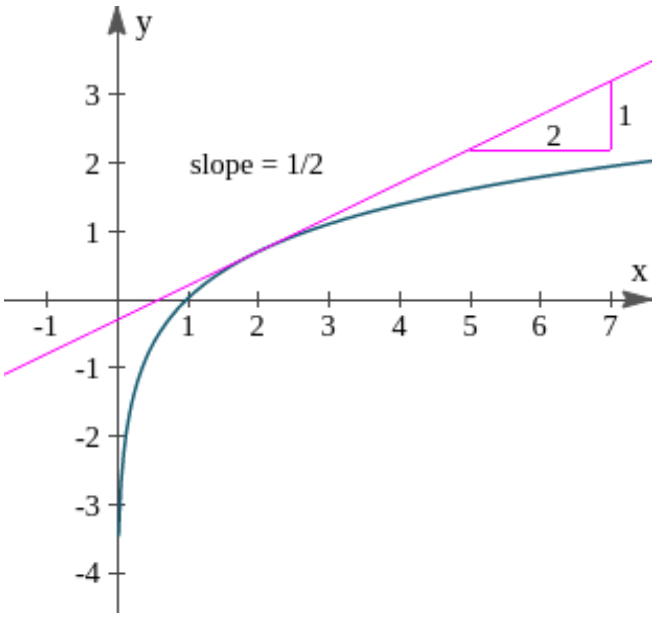

Figure 4-1: Logarithmic function. When using logarithmic losses, as the error approaches zero, the function's gradient increases towards infinity. Image reprinted from [43].

4.1.2.2 Temporal Convolutional Network

We also trained multiple temporal convolutional networks for the task of bounding box regression. We made the inputs to have the 4xn matrix shape where n is the number of frames that their information is used for regression. We also used zero padding to increase the receptive fields. According to our experiments, the best results were achieved when the input filter size was 4x4. It slightly improved the results of multi-layer perceptrons, but the results could still be improved. Our experiment results are written in Table 4.2.

CNN #	# of layers	Input size	L1 Error on Validation Set
1	4	64	0.012
2	5	64	0.011
3	6	64	0.011

Table 4-2: Experiments for training temporal CNNs to predict target bounding box coordinates in the next frames using previous bounding box information. A lower number of layers and smaller inputs led to better results when trained on the GOT10K dataset.

The kernel size we used for the first layer was 4x3, and we slid it over the inputs to let it extract temporal features. Other settings used for training CNNs were the same as MLP training settings.

4.1.2.3 Combined Networks

After observing the ddifferent characteristics of CNN and MLP, we decided to make a six-layer combined network. To design the architecture of this network, we made a four-layer CNN and a six-layer MLP and combined their outputs by taking their average. This resulted in a considerable error drop in the validation set, and the error on the validation set decreased from 0.011 to 0.0106. Motivated by this observation, made further experiments with the new combined networks. Table 4.3 compares different networks L1 error with very few layers trained on the GOT10K dataset:

Network #	# of layers	Input size	L1 Error on Validation Set
1	4	64	0.0117
2	5	64	0.0117
3	6	64	0.0106

Table 4-3: Networks with very few layers. The best performing network among all networks trained is the one with the fewest layers.

4.1.2.4 Why Do Deeper Networks Perform Better?

To explain this phenomenon, we should first note that there are many different target objects in the dataset, including human bodies, cars, trucks, human heads, bikes, moving boxes, etc. Each of those objects moves differently from others and the movements are not linear. For this reason, using deep networks which are better capable of modeling non-linearity are a better choice than models with few layers.

4.1.3 Candidate Selection Algorithm

Most of the hyperparameters in the candidate selection algorithm are in equation (3.2):

$$RS_j = S_j - \frac{(DE_j - 0.1) * (SC_t) * (1 - S_j)}{RW} \tag{3.2}$$

As the mathematical formula shows and according to our experiments, when RW is too high, the effect of the candidate selection algorithm decreases until it gets similar to using multiple templates without any candidate selection. When it is set to be too low, it decreases the performance since the path prediction network is not entirely robust to measurement noise and false detections. A value of 0.1 is subtracted from the distance error. By doing so, the distance error is below 0.1, and the target is relatively in the predicted spot and if the sequence

confidence value is high and the tracker is confident, the candidate selection algorithm increases the reliability score for this candidate. In this way, a candidate with a slightly higher classification score far away from the predicted position is not selected over a candidate with a more reliability score.

4.2 Evaluation on Public Datasets

4.2.1 OTB 100 and UAV20L Dataset Evaluation

This section evaluates our tracker on public datasets to better compare our tracker with other state-of-the-art methods. To this end, we considered two major datasets, OTB [9] and UAV20L [53]. OTB is a short-term tracking dataset while UAV20L is a long-term tracking one. The reason we chose these datasets is because most of the long-term trackers in literature not only evaluate their performance on long-term tracking benchmarks, but also on short term ones as well. We first examine MTTSiam on the OTB 50 and OTB100 [9] dataset. There are 100 different targets, such as cars, trucks, humans, human faces, boxes, etc., in the videos of this dataset. The total number of videos in OTB 100 is fewer than 100 since multiple objects are considered the target in some videos. Table 4.4 provides MTTSiam and other methods' success and precision on that dataset. We also test our tracker on UAV20L [53] dataset which consists of 20 long-term tracking videos. Our tracker achieves high success and precision rates in comparison with other state-of-the-art methods.

Tracker name	SiamFC [39]	SiamFC++ [46]	KYS[45]	Dimp [2]	SiamRPN++ [8]	GradNet [47]
Success ↑	58.7	68.3	69.5	68.6	69.6	63.9
Precision ↑	77.2	91.2	91.0	89.9	91.4	86.1
Tracker name	MDNet [48]	Ocean [49]	SiamDW [50]	ECO [51]	MTTSiam_A (Ours)	MTTSiam_R (Ours)
Success ↑	67.8	67.2	67.4	69.1	67.4	**70.4**
Precision ↑	90.9	90.2	90.5	91.0	88.7	**91.6**

Table 4-4: Different trackers' success and precision on OTB 100 [9]. MTTSiam_A denotes our tracker when the backbone is AlexNet [19], and MTTSiam_R denotes our tracker when the backbone is ResNet 50 [27].

According to Table 4.4, this tracker outperforms many state-of-the-art trackers on the OTB 100 [9] dataset which is a short-term tracking dataset. Please note that OTB100 [9] is a short-term tracking dataset, and our proposed tracker is a long-term one. Hence, although it achieves slightly better success and precision rates, its performance on the UAV20L [53] which is a long-term tracking dataset is much better.

We also evaluated our tracker on UAV20L [53] long-term tracking dataset. This dataset is made of 20 long sequences of tracking different objects filmed with unmanned aerial vehicles. According to our experiments, the improvement for long-term tracking datasets is much higher than short-term tracking datasets. This is because of the use of diverse target appearances and the candidate selection algorithm which reduces tracking reliability score for target distractors. Table 4.5 compares this tracker with other state-of-the-art methods on UAV20L [53] dataset and Table 4.6 provides a detailed comparison with the baseline on UAV20L [53] dataset.

Tracker name	SiamFC [39]	HiFT [55]	DaSiamRPN [54]	ECO [51]	SiamRPN++ [8]	MTTSiam (Ours)
Success ↑	40.2	56.6	46.5	58.9	52.8	**65.4**
Precision ↑	59.9	76.3	66.5	42.7	69.6	**86.2**

Table 4-5: Results of comparison on UAV20L [53] long-term tracking dataset.

Overlap Threshold	0.8	0.6	0.4
MTTSiam (Ours)	**48.1**	**77.3**	**83.1**
SiamRPN++	40.0	62.3	66.7

Table 4-6: Success rates of OPE on UAV20L [53] dataset at different overlap thresholds.

MTTSiam has a much higher success rate at different thresholds.

4.2.2 Ablation Study on OTB 50

Table 4.7 provides the results of the ablation study over the OTB50 dataset. According to our experiments, the candidate selection algorithm performs best when combined with the template updating process. This Table also shows the effect of each of our contributions equally. According to this Table, using multiple templates and changing them dynamically has a higher effect on achieving better results.

Tracker name	SiamRPN++ (AlexNet)	SiamRPN++ (ResNet50)	MTTSiam (AlexNet) (ours)	Tsiam (AlexNet) (ours)	MTSiam (AlexNet) (ours)	MTTSiam (ResNet50) (ours)
Success ↑	61.7	66.5	64.2	61.9	63.3	**66.8**
Precision ↑	83.3	88.6	87.6	83.9	85.7	**89.5**

Table 4-7: Ablation study on OTB50 dataset. TSiam means using only the candidate selection algorithm over the baseline. MTSiam denotes using multiple templates without the candidate selection algorithm.

4.2.3 Experiments Outside Datasets

In order to provide a comparison between our tracker and the baseline we took a video by a phone camera. Since dataset success and precision rates are not always the best metrics, we evaluated this tracker on a real-life video. Figure 4.3 shows the results of the comparison between our tracker MTTSiam with the baseline. The video image resolution is 4160x3120. The purpose is to track the right hand despite its change in appearance and presence of similar objects to the target in the background. In that experiment both trackers use the AlexNet [19] as their backbone. MTTSiam successfully adapts itself with the changes, unlike SiamRPN++ which tracks the forehead instead of the hand at some points.

Figure 4-2: Comparison of the MTTSiam with SiamRPN++ tracker. Both trackers have the same initialization and try to track the right hand. The row above shows the results of MTTSiam whereas the row below shows the results for the SiamRPN++ tracker [8].

In the following parts some Figures are added to this thesis to provide qualitative results on how this tracker performs on the datasets videos. The OTB 100 [9] dataset is quite challenging since it has many blurred frames of targets, many partial and full occlusion cases of the target, change of illumination, pose variation and combination of different challenging scenarios at once. Figure 4.3 shows an example of tracking in blurry frames.

Figure 4-3: Tracking in blurry videos. The model needs to be able to adapt itself to such challenges. Video frames were taken from a video sequence in the OTB 100 [9] dataset. The green line shows the ground truth, and the yellow lines show the performance of the MTTSiam tracker.

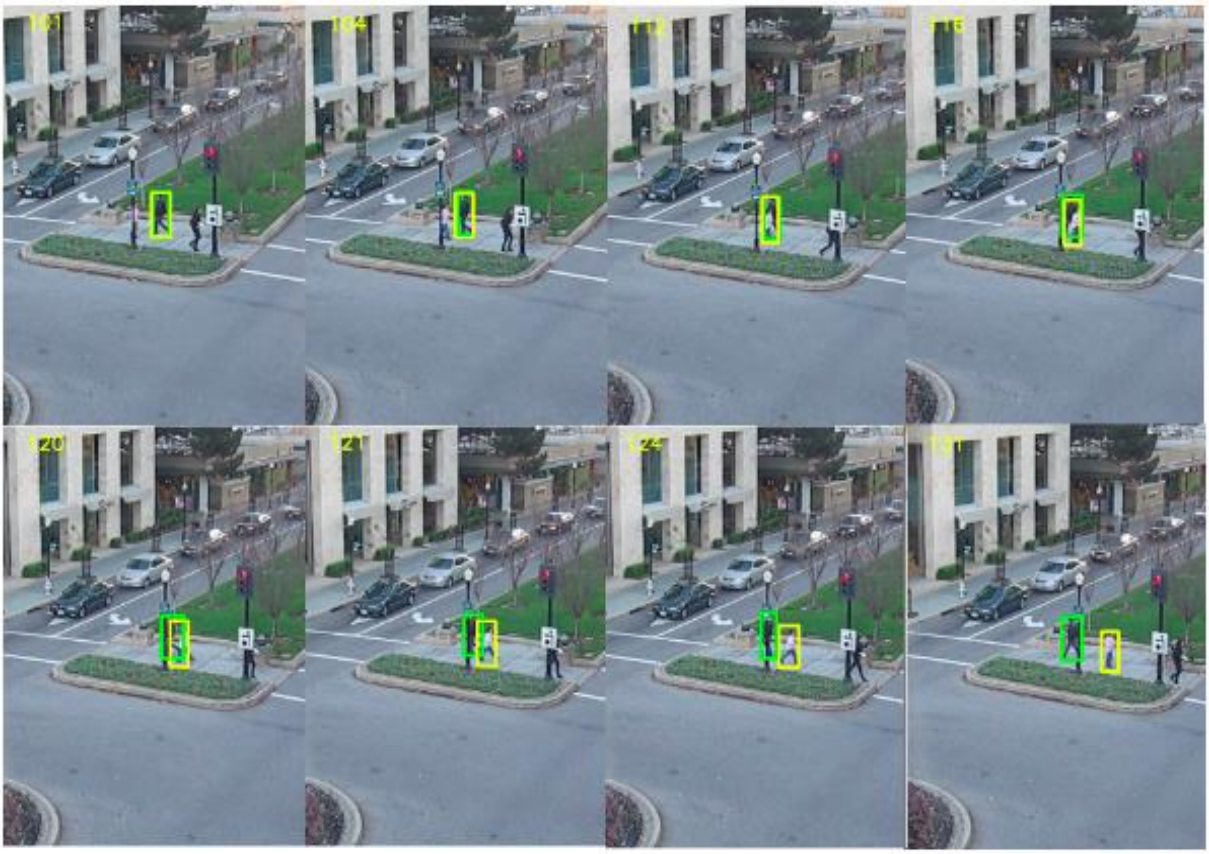

Figure 4-4: When the target is occluded with an object of the same class, the tracker mistakenly tracks the distractor instead of the original target. After the woman passes in front of the man, the tracker tracks her instead of him in this Figure. The image sequence is from the OTB 100 [9] dataset. The green bounding box shows the ground-truth label, and the yellow bounding box shows MTTSiam tracker results. The network backbone in this image is AlexNet. This problem can be addressed by using better backbones.

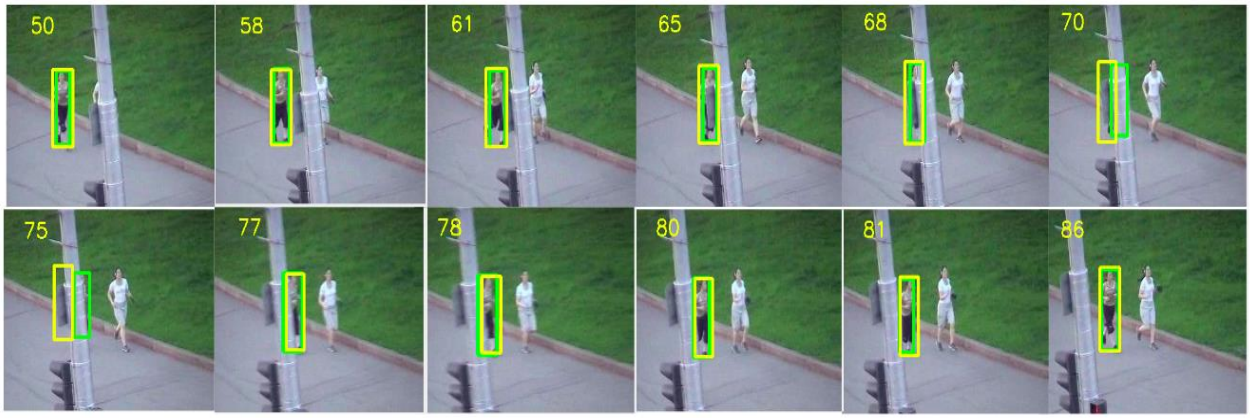

Figure 4-5: Full occlusion for a short time with the presence of a similar object to the target in the background. After the occlusion is finished, the target is re-identified, and tracking continues. The green bounding box is the annotated label, and the yellow bounding box shows the MTTSiam tracker results. The image sequence is from the OTB 100 [9] dataset.

4.3 Speed

In this section we will have an overview of what normally affects the speed of a tracker, and then we will compare our tracker with its baseline and other state-of-the-art trackers in terms of speed.

When tracking an object in a video, depending on the tracker architecture many things can affect the tracking speed. For instance, some trackers rely on correlation to find the target in an image. In this case, the bigger the filter, the slower the tracker. Most of the short-term trackers try to find the target around where it has last been seen, by searching an area centered where the target has been last seen. The search window must be bigger than the target size. As a result, when the target is bigger, in most cases it increases computation complexity. Another factor is image resolution. High resolution images require more processing. In some long-term trackers, when tracking score is above a certain threshold, the search window is an area slightly bigger than the target size. But when the score is below that threshold, the search window is

the whole image. As a result, when tracking score is high the tracker performs much faster than when the score is low. Tracking speed also always relies on the speed of the processing unit i.e., CPU or GPU as well.

With all that being said, we evaluate the speed of our tracker on the OTB 100 [9] dataset. The processing unit used in our experiments is a single GPU RTX 3080. When using AlexNet [19] as the backbone, the speed is 58 frapes per second (fps). When ResNet 50 [27] is used, the speed is reduced to 18 fps. This is almost as half fast as the baseline, SiamRPN++ [8] tracker.

Chapter 5. Conclusion

Long-term object tracking is more similar to how object tracking is done in real-life with more practical applications. Most of the state-of-the-art long-term trackers are based on the Siamese architecture where the target appearance in the first frame is used to localize it in given search frames. Such works usually are not robust against target appearance changes or distinguishing targets from background distractors. This study proposes a new long-term tracker based on the Siamese architecture that uses a bag of templates to keep track of target appearance changes. Furthermore, most detection-based trackers suffer from switching between the real target and possible distractors. In order to take care of this problem we propose the temporal network which predicts target location given its bounding box coordinates in

previous frames. The temporal network predictions are used to select some target candidates over others in the candidate selection algorithm. The result of the ablation study is provided in Table 4.7 on the OTB 50 dataset. To the best of our knowledge, we are the first deep learning approach to predict target bounding box coordinates given its coordinates in previous frames.

5.1 Accomplishments

We propose a new multi-template Siamese architecture for robust object tracking with new strategies for template replacement for short-term and long-term tracking. Combined with the temporal network and candidate selection algorithm it achieves leading results in OTB 100 [9] short-term dataset and UAV20L [53] long-term datasets. We also evaluate our tracker on real-life videos and compare its performance with the baseline to provide more insight into the effect of changes we made in this work.

5.2 Contributions

In short, in this study, we had three main contributions:

- **Multi-template Siamese feature extractor** uses target appearance in different frames to localize it in the search image. We proposed new strategies for replacing short-term and long-term tracking templates, leading to high-performance tracking.
- A **temporal network** that learns the history of target trajectory, size, and shapes only having its bounding box coordinates and makes predictions on where the target can most probably be found in the search image.
- A new **Candidate selection algorithm** which combines the path information with candidates' positions and classification scores and the tracker confidence history, providing robust tracking with high target reliability scores.

5.3 Future Work

In this work we observed the effect of template updating and temporal information and how it enhances long-term tracking. Nowadays a recent trend in computer vision tasks is the use of transformers in images and they get more and more widely spread every day. Many recent trackers such as HiFT [55] with a relatively similar architecture to the previous trackers and with the use of transformers obtain highly sophisticated results on different tracking benchmarks. We believe that by using transformers in this work, replacing them with the RPN heads and using attention, this tracker can be greatly enhanced to obtain even better results.

Chapter 6. References

[1] Bolme, D.S., Beveridge, J.R., Draper, B.A. and Lui, Y.M., 2010, June. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition* (pp. 2544-2550). IEEE.

[2] Bhat, G., Danelljan, M., Gool, L.V. and Timofte, R., 2019. Learning discriminative model prediction for tracking. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 6182-6191).

[3] Voigtlaender, P., Luiten, J., Torr, P.H. and Leibe, B., 2020. Siam r-cnn: Visual tracking by re-detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6578-6588).

[4] He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 2961-2969).

[5] Luiten, J., Voigtlaender, P. and Leibe, B., 2018, December. Premvos: Proposal-generation, refinement and merging for video object segmentation. In *Asian Conference on Computer Vision* (pp. 565-580). Springer, Cham.

[6] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E. and Shah, R., 1993. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, *6*.

[7] Schroff, F., Kalenichenko, D. and Philbin, J., 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 815-823).

[8] Li, B., Wu, W., Wang, Q., Zhang, F., Xing, J. and Yan, J., 2019. Siamrpn++: Evolution of siamese visual tracking with very deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4282-4291).

[9] Y. Wu, J. Lim, and M.-H. Yang. Object tracking benchmark. TPAMI, 2015

[10] Al-Shamisi, M.H., Assi, A.H. and Hejase, H.A., 2013. Artificial neural networks for predicting global solar radiation in Al Ain city-UAE. *International journal of green energy*, *10*(5), pp.443-456.

[11] Holmgren, G., Andersson, P., Jakobsson, A. and Frigyesi, A., 2019. Artificial neural networks improve and simplify intensive care mortality prognostication: a national cohort study of 217,289 first-time intensive care unit admissions. *Journal of intensive care*, *7*(1), pp.1-8.

[12] Dahl, G.E., Sainath, T.N. and Hinton, G.E., 2013, May. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 8609-8613). IEEE.

[13] Keldenich, T., 2022. *Activation function, how does it work? - A simple explanation*. [online] Inside Machine Learning. Available at: <https://inside-machinelearning.com/en/activation-function-how-does-it-work-a-simple-explanation> [Accessed 1 June 2022].

[14] i2tutorials. 2022. *Explain Step / Threshold and Leaky ReLU Activation Functions | i2tutorials*. [online] Available at: <https://www.i2tutorials.com/explain-step-threshold-and-leaky-relu-activation-functions/> [Accessed 1 June 2022].

[15] Commons.wikimedia.org. 2022. *File:Sigmoid-function-2.svg - Wikimedia Commons*. [online] Available at: <https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg> [Accessed 1 June 2022].

[16] Li, B., Yan, J., Wu, W., Zhu, Z. and Hu, X., 2018. High performance visual tracking with siamese region proposal network. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8971-8980).

[17] Datathings.com. 2022. *Neural networks and backpropagation explained in a simple way · Datathings' Blog*. [online] Available at: <https://datathings.com/blog/post/neuralnet/> [Accessed 2 June 2022].

[18] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *nature*, *521*(7553), pp.436-444.

[19] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

[20] Brownlee, J., 2022. *How Do Convolutional Layers Work in Deep Learning Neural Networks?*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/> [Accessed 1 June 2022].

[21] Chuah, W., 2022. *Learning Convolutional Neural Network (CNN) with PyTorch*. [online] Medium. Available at: <https://python.plainenglish.io/learning-convolutional-neural-network-cnn-with-pytorch-b10753898130> [Accessed 1 June 2022].

[22] Murray, N. and Perronnin, F., 2014. Generalized max pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2473-2480).

[23] DeepAI. 2022. *Max Pooling*. [online] Available at: <https://deepai.org/machine-learning-glossary-and-terms/max-pooling> [Accessed 1 June 2022].

[24] Moon, C., 2022. *L1 and L2 as Regularization for a Linear Model - Things to Know about Machine Learning*. [online] Things to Know about Machine Learning. Available at: <http://csmoon-ml.com/index.php/2018/12/20/l1-and-l2-as-regularization-for-a-linear-model/> [Accessed 1 June 2022].

[25] Chen, T., Kornblith, S., Norouzi, M. and Hinton, G., 2020, November. A simple framework for contrastive learning of visual representations. In *International conference on machine learning* (pp. 1597-1607). PMLR.

[26] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), pp.1929-1958.

[27] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[28] Sinha, U., 2022. *The Sobel and Laplacian Edge Detectors - AI Shack*. [online] Aishack.in. Available at: <https://aishack.in/tutorials/sobel-laplacian-edge-detectors> [Accessed 1 June 2022].

[29] Harris, C. and Stephens, M., 1988, August. A combined corner and edge detector. In *Alvey vision conference* (Vol. 15, No. 50, pp. 10-5244).

[30] Veenman, C.J., Reinders, M.J. and Backer, E., 2001. Resolving motion correspondence for densely moving points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(1), pp.54-72.

[31] Lowe, D.G., 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), pp.91-110.

[32] Henriques, J.F., Caseiro, R., Martins, P. and Batista, J., 2014. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence*, *37*(3), pp.583-596.

[33] Lukezic, A., Vojir, T., ˇCehovin Zajc, L., Matas, J. and Kristan, M., 2017. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 6309-6318).

[34] Sekhavati, A. and Eghbal, N., 2020. Auto-correct-integrated trackers with and without memory of first frames. *International Journal of Intelligent Robotics and Applications*, *4*(2), pp.191-201.

[35] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, *28*.

[36] Tan, M., Pang, R. and Le, Q.V., 2020. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10781-10790).

[37] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[38] Tan, M. and Le, Q., 2019, May. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.

[39] Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A. and Torr, P.H., 2016, October. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision* (pp. 850-865). Springer, Cham.

[40] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[41] Rastogi, A., 2022. *ResNet50*. [online] Medium. Available at: <https://blog.devgenius.io/resnet50-6b42934db431> [Accessed 1 June 2022].

[42] Li, C., Xing, Q., Ma, Z. and Zang, K., 2020. MFCFSiam: A Correlation-Filter-Guided Siamese Network with Multifeature for Visual Tracking. *Wireless Communications and Mobile Computing*, *2020*.

[43] Bourne, M., 2022. *Differentiation of Transcendental Functions*. [online] Intmath.com. Available at: <https://www.intmath.com/differentiation-transcendental/> [Accessed 1 June 2022].

[44] Real, E., Shlens, J., Mazzocchi, S., Pan, X. and Vanhoucke, V., 2017. Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5296-5305).

[45] Bhat, G., Danelljan, M., Gool, L.V. and Timofte, R., 2020, August. Know your surroundings: Exploiting scene information for object tracking. In *European Conference on Computer Vision* (pp. 205-221). Springer, Cham.

[46] Xu, Y., Wang, Z., Li, Z., Yuan, Y. and Yu, G., 2020, April. Siamfc++: Towards robust and accurate visual tracking with target estimation guidelines.

In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 07, pp. 12549-12556).

[47] Li, P., Chen, B., Ouyang, W., Wang, D., Yang, X. and Lu, H., 2019. Gradnet: Gradient-guided network for visual object tracking. In *Proceedings of the IEEE/CVF International conference on computer vision* (pp. 6162-6171).

[48] Nam, H. and Han, B., 2016. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4293-4302).

[49] Zhang, Z., Peng, H., Fu, J., Li, B. and Hu, W., 2020, August. Ocean: Object-aware anchor-free tracking. In *European Conference on Computer Vision* (pp. 771-787). Springer, Cham.

[50] Zhang, Z. and Peng, H., 2019. Deeper and wider siamese networks for real-time visual tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4591-4600).

[51] Danelljan, M., Bhat, G., Shahbaz Khan, F. and Felsberg, M., 2017. Eco: Efficient convolution operators for tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 6638-6646).

[52] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C., 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), pp.211-252.

[53] Mueller, M., Smith, N. and Ghanem, B., 2016, October. A benchmark and simulator for uav tracking. In *European conference on computer vision* (pp. 445-461). Springer, Cham.

[54] Zhu, Z., Wang, Q., Li, B., Wu, W., Yan, J. and Hu, W., 2018. Distractor-aware siamese networks for visual object tracking. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 101-117).

[55] Cao, Z., Fu, C., Ye, J., Li, B. and Li, Y., 2021. HiFT: Hierarchical Feature Transformer for Aerial Tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 15457-15466).

[56] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.

[57] M. Kristan et al., "The Ninth Visual Object Tracking VOT2021 Challenge Results," 2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2021, pp. 2711-2738.

[58] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.