ATTENTIVE DEEP REGRESSION NETWORKS FOR REAL-TIME VISUAL FACE TRACKING IN VIDEO SURVEILLANCE

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

BY

SAFA ALVER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JULY 2019

Approval of the thesis:

ATTENTIVE DEEP REGRESSION NETWORKS FOR REAL-TIME VISUAL FACE TRACKING IN VIDEO SURVEILLANCE

submitted by **SAFA ALVER** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Ilkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** ________________

Prof. Dr. Uğur Halıcı
Supervisor, **Electrical and Electronics Engineering, METU** ________________

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Department, METU ________________

Prof. Dr. Uğur Halıcı
Electrical and Electronics Engineering Department, METU ________________

Prof. Dr. Alptekin Temizel
Modelling and Simulation Department, METU ________________

Assist. Prof. Dr. Elif Vural
Electrical and Electronics Engineering Department, METU ________________

Assist. Prof. Dr. Tolga İnan
Electrical and Electronics Engineering Department, Çankaya Uni. ________________

Date:

ABSTRACT

ATTENTIVE DEEP REGRESSION NETWORKS FOR REAL-TIME VISUAL FACE TRACKING IN VIDEO SURVEILLANCE

Alver, Safa
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Uğur Halıcı

July 2019, 113 pages

Visual face tracking is one of the most important tasks in video surveillance systems. However, due to the variations in pose, scale, expression and illumination and the occlusions in cluttered scenes, it is considered to be a difficult task. To address these challenges, in this thesis, we propose an end-to-end tracker named Attentive Face Tracking Network (AFTN) that is build on top of the GOTURN tracker. Additionally, to overcome the scarce data problem in visual face tracking, we also provide bounding box annotations for the publicly available ChokePoint dataset and thus make it available for further studies in face tracking under surveillance conditions. Our test results show that our proposed tracker outperforms all the other trackers that are primitive versions of itself. Furthermore, it runs at speeds that are far beyond the requirements of real-time tracking.

Keywords: Channel Attention, Convolutional Neural Networks, Deep Learning, Video Surveillance, Visual Face Tracking, Visual Object Tracking

ÖZ

VİDEOLU GÖZETİMDE GERÇEK ZAMANLI GÖRSEL YÜZ TAKİBİ İÇİN DİKKAT ODAKLAMALI DERİN REGRESYON AĞLARI

Alver, Safa
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Uğur Halıcı

Temmuz 2019 , 113 sayfa

Videolu gözetim sistemlerinde gerçekleştirilen en önemli işlerden birisi görsel yüz takibidir. Ancak poz, ölçek, ifade ve aydınlatmadaki değişiklikler ve karışık sahnelerdeki kapanmalar nedeniyle, zor bir iş olarak kabul edilir. Bu zorlukların üstesinden gelmek için, bu tez çalışmasında, GOTURN takipçisinin üzerine inşa edilen Dikkat Odaklamalı Yüz Takip Ağı (DOYTA) adlı uçtan uca bir takipçi öneriyoruz. Ek olarak, görsel yüz takibinde mevcut olan yetersiz veri probleminin üstesinden gelmek için kamuya açık ChokePoint veri kümesinin sınırlayacı kutu açıklamalarını sağlıyoruz ve böylece gözetim koşulları altında yüz takibi konusunda daha ileri çalışmalar için kullanılabilir hale getiriyoruz. Test sonuçlarımız, önerilen takipçimizin, ilkel sürümleri olan diğer tüm takipçileri geride bıraktığını gösteriyor. Ayrıca, gerçek zamanlı takip gereksinimlerinin çok ötesinde olan hızlarda çalışmaktadır.

Anahtar Kelimeler: Kanal Odaklaması, Evrişimsel Sinir Ağları, Derin Öğrenme, Video Gözetimi, Görsel Yüz Takibi, Görsel Nesne Takibi

To my beloved family and beautiful country...

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Prof. Dr. Uğur Halıcı for her support, guidance, motivation and patience throughout this thesis study. Her easygoing attitude has helped me a lot in exploring the different areas of machine learning during my M.Sc. studies.

But must of all, I would like to express my gratitude to my family for their never-ending support and the freedom they gave me to pursue my dreams. Thank you. My father, Prof. Dr. Ümit Alver, also probably deserves a co-supervision credit for the many hours he spent listening to my ideas and for the inspiration he has been to me throughout my whole life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D	1 Dimensional
2D	2 Dimensional
AAC	Area Above Curve
Adam	Adaptive Moment Estimation
AUC	Area Under Curve
CAN	Channel Attention Network
CNN	Convolutional Neural Network
FC	Fully Connected
FAN	Feature Adaptation Network
FEN	Feature Extraction Network
FR	Failure Rate
GOTURN	Generic Object Tracking Using Regression Networks
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
NN	Neural Network
RMS	Root Mean Square
RN	Regression Network
RNN	Recurrent Neural Network
ROT	Region Overlap Threshold
RT	Reinitialization Threshold
SGD	Stochastic Gradient Descent
TP	True Positive
VGG	Visual Geometry Group
VOT	Visual Object Tracking

CHAPTER 1

INTRODUCTION

Video surveillance systems are widely deployed in both public and private places for the purpose of verifying or recognizing the individuals of interest. However, most of these systems are monitored by human operators and thus have problems in reliability and scalability. Hence, an automated approach in security monitoring is required. In these automated approaches, the faces of the individuals have to be first detected and then tracked before the higher level recognition or verification tasks. Thus face tracking is one of the crucial elements in automated video surveillance systems. Due to the variations in pose, scale, expression and illumination and the occlusions in cluttered scenes, it is considered to be a difficult task by itself.

While the current learning-based video surveillance face tracking systems [1] perform up to certain degrees, they cannot run at speeds that are required (>25 FPS) for real-time tracking. Additionally, these learning-based methods cannot avoid the drifting problem caused by learning the background or occlusion. Although a color histogram-assisted face tracker proposed in [2] can perform real-time tracking (>100 FPS), it makes use of the semantically weak color information where any similarly colored distractor around the face can lead to tracking the wrong target. Importantly, none of the face trackers in these studies make use of the learned rich hierarchical features that can serve very well in representing and thus tracking human faces under various harsh conditions.

Most recently, deep learning based approaches has yielded significant performance increase in a wide variety of computer vision tasks as image classification [3–5], object detection [6], semantic segmentation [7–9], image question answering [10], face verification [11], pose estimation [12], as well as object tracking [13–15]. Such great successes of deep learning based approaches is attributed mostly to their generalization capability, representation power and topological biases to exploit the structure of the data. These properties make them very suitable and adaptable for a wide variety of problems both in computer vision and other fields as natural language processing and reinforcement learning.

Despite the difficulties and challenges present in face tracking and object tracking in general, these deep learning based approaches have also shown state-of-the-art results in the recent visual object tracking (VOT) challenges[1]. Starting from 2015, the deep learning based trackers as MDNet [13], TCNN [16], SiamFC [15] and SiamRPN [17] have performed among the top in the VOT challenges [18–21]. Further, the ones that are trained in an offline manner can reach speeds that surpass the real-time tracking requirements.

Motivated by this, we propose an end-to-end deep learning based method for the task of visual face tracking under surveillance conditions. Evaluation results show that our tracker outperforms all of the other trackers where the GOTURN tracker [14] is used as a baseline. It also outperforms one of the best [1] surveillance face tracker named Incremental Visual Tracking (IVT) [22] by a very large margin. Furthermore, it runs at speeds ($\sim$140 FPS or $\sim$180 FPS) that are very far beyond the requirements of real-time tracking.

1.1 Problem Definition

In this thesis, we address the problem of face tracking under surveillance conditions where the input is a sequence of raw video frames and the output is a tuple that describes the location and size of the target's bounding box for each of these frames. More specifically, given the bounding box of a face in the first frame, the objective of a tracker is to report the bounding box annotations in all of the following frames in the video sequence. Different from the task of face detection where the face is located independently in each frame, in face tracking, the previous location information is

[1] The VOT challenges provide the visual object tracking community with a precisely defined and repeatable way of comparing short-term trackers as well as a common platform for discussing the evaluation and advancements made in the field of visual tracking.

also used as a prior for localizing the target. A good face tracker must not only keep track of the face as long as they is present in the video frame, but it must also do it well, i.e., the tracker's predictions should match closely to the ground-truth annotations since small errors in each frame can easily lead to a target loss.

1.2 Contributions

The main contributions of this thesis is as follows:

- **Investigation of an end-to-end deep learning method for the task of visual face tracking under surveillance conditions:** We showed that an improved end-to-end deep learning based generic object tracker that directly regresses the bounding box annotations from raw video frames can be trained to track faces under surveillance conditions. A thorough search of the relevant literature had yielded no published study on using deep learning methods that are trained in a fully end-to-end manner for the task of visual face tracking in video surveillance.
- **An improved attentive network for real-time single-target visual object tracking:** We took the real-time single-target GOTURN tracker [14] as our baseline and improved it using several useful extensions as using a fusion network in the regression network, making use of the lower level features and using a channel-wise attention mechanism for adaptive channel selection. Although we have used this network to track faces, it can be used for any real-time single-target visual object tracking task without further modification in the architecture. It just needs to be trained in an offline manner with the domain specific dataset.
- **Bounding box annotations for the ChokePoint dataset:** We provided accurate bounding box annotations for the G1 and G2 sets of the publicly available ChokePoint dataset [23] and thus made it available for further studies in visual face tracking under surveillance conditions. The original dataset only has person ID and eye location annotations which are not compatible with the task of visual tracking.

1.3 Outline

The rest of this thesis is organized as follows:

- **Chapter 2** presents a historical, practical and theoretical background in neural networks and deep learning that is essential in understanding the work done in this thesis. After a brief overview of machine learning and neural network history, it discusses vanilla neural networks and their optimization methods. It then moves on to more complicated models called convolutional neural networks. Finally, it presents some of the common practical tricks that are used in neural network training and moves on to transfer learning.
- **Chapter 3** starts by presenting the traditional and deep learning based methods for the general task of single-target visual object tracking and then moves on to the studies that incorporate additional attention mechanisms for adaptive channel selection. Afterwards, it goes over the studies on the specific task of visual face tracking under surveillance conditions. Lastly, it discusses the similarities and differences between our proposed face tracker with each of the presented studies.
- **Chapter 4** starts by describing the details of the proposed network architecture and then moves on to the details of the employed channel-wise attention mechanism. Afterwards, it presents the details of the offline training and online tracking procedures.
- **Chapter 5** starts by describing the face dataset that is used in this thesis. It then moves on to the details of the evaluation metrics that are used in comparing the test trackers. After the description of the evaluation metrics, it continues by presenting the details of the test network architectures together with their testing procedures. It then presents the quantitative accuracy, robustness and speed results using several informative plots and tables. Lastly, it provides a qualitative analysis to visualize the effect of the channel-wise attention mechanism.
- **Chapter 6** concludes the main contributions of this thesis and discusses the limitations and some of the possible future directions that can be further investigated.

CHAPTER 2

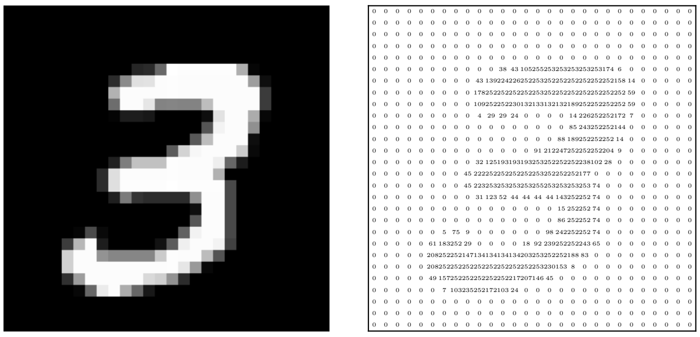
BACKGROUND ON DEEP LEARNING

This chapter provides a historical, practical and theoretical background in neural networks and deep learning. After giving a brief overview of machine learning and neural network history, vanilla neural networks and their optimization methods will be discussed. Then more complicated models, which are frequently used in computer vision, called convolutional neural networks and their arithmetic will be explained. Finally, some practical tricks for training these networks and transfer learning methods will be presented.

2.1 A Brief Overview of Machine Learning

Machine learning is a subfield of Artificial Intelligence (AI) which studies the algorithms and statistical models that computers use to *learn* specific tasks from experience and without being explicitly programmed. These tasks usually require some level of intelligence and are very hard to solve by rule-based programs. Machine learning algorithms build mathematical models from the training data to make predictions or decisions on unseen data to solve these hard tasks. Visual recognition is one of these hard tasks. For instance, although it is obvious for a human being that the MNIST [24] image on the left side of Figure 2.1 is a 3, what the computer sees is just a bunch of integers on a grid as in the right side of it. Writing a rule-based program to recognize all types of hand-written 3 images is nearly impossible and this is where the statistical models of machine learning come in.

Based on the information available during training, machine learning algorithms can be grouped into three main categories:



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	38	43	105	255	253	253	253	253	253	174	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	43	139	224	226	252	253	252	252	252	252	252	252	158	14	0	0	0	0	0
0	0	0	0	0	0	0	0	0	178	252	252	252	252	253	252	252	252	252	252	252	252	59	0	0	0	0	0
0	0	0	0	0	0	0	0	0	109	252	252	230	132	133	132	132	189	252	252	252	252	59	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4	29	29	24	0	0	0	0	14	226	252	252	172	7	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	85	243	252	252	144	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	189	252	252	252	14	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	91	212	247	252	252	252	204	9	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	32	125	193	193	193	253	252	252	252	238	102	28	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	45	222	252	252	252	252	253	252	252	252	177	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	45	223	253	253	253	253	255	253	253	253	253	74	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	31	123	52	44	44	44	44	143	252	252	74	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	252	252	74	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86	252	252	74	0	0	0	0	0	0	0	0
0	0	0	0	0	0	5	75	9	0	0	0	0	0	0	98	242	252	252	74	0	0	0	0	0	0	0	0
0	0	0	0	0	61	183	252	29	0	0	0	0	18	92	239	252	252	243	65	0	0	0	0	0	0	0	0
0	0	0	0	0	208	252	252	147	134	134	134	134	203	253	252	252	188	83	0	0	0	0	0	0	0	0	0
0	0	0	0	0	208	252	252	252	252	252	252	252	252	253	230	153	8	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	49	157	252	252	252	252	252	217	207	146	45	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	7	103	235	252	172	103	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.1: Left: A random 28×28 hand-written image of 3 from the famous MNIST dataset. Right: The same image that is viewed by a computer which is just a bunch of numbers on a grid.

- **Supervised Learning:** In supervised learning, training data comes with labels, i.e., for each training example there is a corresponding label and the goal is to learn a function that maps the training examples to these labels. Object detection/tracking/classification are common examples of supervised learning in which the image is the training example and the bounding box / object class is the label.
- **Unsupervised Learning:** In unsupervised learning, training data has no labels and the goal is to model the data distribution. Clustering and dimensionality reduction are among the common examples of unsupervised learning.
- **Reinforcement Learning:** In reinforcement learning, training data comes with a numerical reward signal instead of a direct label. Specifically, the training data and reward comes from the interaction of the agent with its environment and the goal is to learn a policy that maximizes the expected cumulative reward. Learning to play games and learning to drive cars are among the common examples of reinforcement learning. In fact, many of the sequential decision problems can be formulated in a reinforcement learning setting.

Although they can be incorporated in several ways, unsupervised learning and reinforcement learning are out of the scope of this study. In this thesis, only the supervised learning paradigm will be used to perform one of the hard tasks described above. More specifically, parametric machine learning models inspired by mammalian brains called neural networks will be used to perform the visual face tracking task.

Two of the most popular tasks performed in supervised learning are classification and regression:

- **Classification:** Classification is the task of predicting the class label of a given data. The machine learning model acts as a function $f(.)$ mapping n dimensional inputs to a discrete space of k categories, i.e., $f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$. Given a dataset containing images of cats and dogs, predicting which image has a dog or a cat can be thought of a classification task.
- **Regression:** Regression on the other hand, is the task of predicting real-valued labels of a given data. In these tasks, the machine learning model acts as a function $f(.)$ mapping n dimensional inputs to m dimensional outputs, i.e., $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Predicting the bounding box annotations of the cats and dogs in a given cat-dog dataset is an example of a regression task.

This section provided a brief overview of machine learning. However, it is by no means a formal and in-depth view to the topic. Interested readers can find more information in the Machine Learning book of Bishop [25] and in Part I of the Deep Learning book of Goodfellow et al. [26].

2.2 A Brief History of Neural Networks

Although the history of neural networks can be dated back at least to the 1800s studies of Legendre [27] and Gauss [28, 29] on linear regression, these studies were not explicitly trying to model the neurons in the brain. Thus, this section will start with the artificial neuron.

The artificial neuron is a simplified mathematical model of the biological neuron in animal brains (see Figure 2.2) that is proposed by McCulloch and Pitts [30] in 1943. The artificial neuron proposed in their study was able to solve some simple binary problems by using preset parameters; however, it was not able to adapt its parameters towards a specific goal, i.e., it was not able to learn. The first (unsupervised) learning

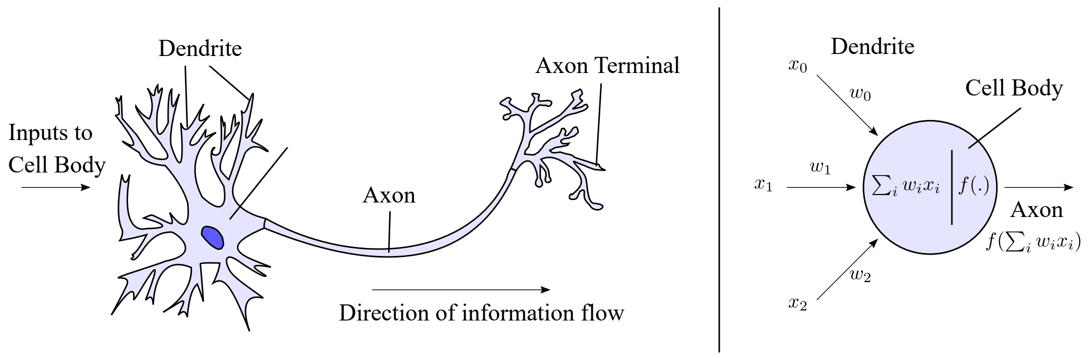


Figure 2.2: Left: Simplified drawing of a biological neuron. Right: The artificial neuron which is a simplified model of the biological neuron. The weights w_i of the artificial neuron models the synaptic connection strengths between axon terminals and dendrites of the biological neuron.

rule was proposed by Hebb in 1949 [31] which simply strengthened the connection between two neurons if they fired simultaneously and weakened otherwise. However, its requirement on input orthogonality placed serious limitations on it. After nearly a decade, in 1958, Hebb's learning rule inspired Rosenblatt to come up with the Perceptron [32] which was the first artificial neuron that was able to learn (by supervised learning) the parameters required to output a desired outcome.

The perceptron first takes a weighted sum of each dimension of an n dimensional input vector $\boldsymbol{x} = (x_0, \ldots, x_n)$ with its corresponding weight vector $\boldsymbol{w} = (w_0, \ldots, w_n)$ to compute z as in equation 2.1. Note the simplified notation where x_0 is set to 1 and the bias term b is replaced by w_o, i.e., $x_0 = 1$ and $w_0 = b$. The weighted sum can also be written as a dot product between $\boldsymbol{w}$ and $\boldsymbol{x}$ as in equation 2.1.

$$z = \sum_{i=1}^{n} w_i x_i + b = \sum_{i=0}^{n} w_i x_i = \boldsymbol{w}^T \cdot \boldsymbol{x} \tag{2.1}$$

It then passes this weighted sum z through a step activation function to obtain the

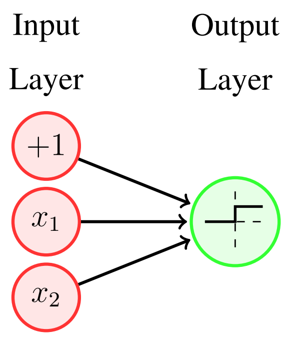


Figure 2.3: A perceptron with two inputs.

binary output y as in equation 2.2.

$$y = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

Although the preceptron was initially used as a binary classifier, it can also be extended to multiple classes by simply adding more dimensions to the output y. An example perceptron with two inputs is given in Figure 2.3.

The most remarkable contribution of the perceptron was perhaps the learning algorithm it introduced that was able to learn from data. Given a perceptron with parameters $\boldsymbol{w}$ and a training data pair $(\boldsymbol{x}, y)$, the parameter update rule of the perceptron is as in equation 2.3,

$$\boldsymbol{w} := \boldsymbol{w} - \eta(\hat{y} - y)\boldsymbol{x} \tag{2.3}$$

where y is the desired output, $\hat{y}$ is the perceptron output and η is the learning rate.

With the introduction of the perceptron in 1958, a new era in neural network research has started. However, approximately a decade later, in 1969, Minsky and Papert published a book called “Perceptrons” [33] which showed that the perceptron was only able to solve linearly separable problems and failed at learning non-linearly separable ones including basic logical binary functions as XOR (see Figure 2.4). Although, they showed that a perceptron with intermediate layers could solve the XOR problem, they stated that the learning algorithm of the perceptron is limited to perceptrons with

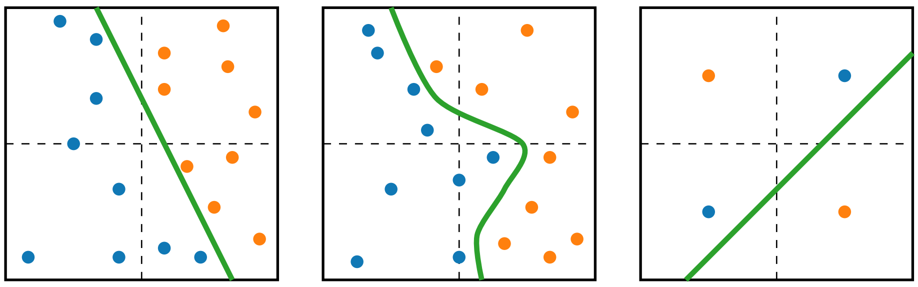


Figure 2.4: Left: A linearly separable problem. Middle: A non-linearly separable problem. Right: The XOR problem that cannot be solved by a linear line. At least one hidden layer is required to solve it.

just a single layer. With this book, lots of the researchers working on neural networks stopped working on them and research funds were cut which led to the era commonly known as the *first AI winter*. Although it should be noted that Ivakhnenko, known by some as the “Father of Deep Learning” [34], was already performing learning in large multilayer perceptron-like networks in 1968 by a method named group method data handling and its code was shared [35, 36].

The first AI winter lasted approximately a decade and in the early 1980s research on neural networks gained popularity again. In 1982, Hopfield presented a recurrent neural network with associative memory that was useful in both understanding the working mechanism of the brain and in practical applications [37]. With the 1986 paper of Rumelhart et al. [38] showing that neural networks trained with backpropagation can generate useful representations, the interest in the field was restored. However, the expectations were set too high and shortly thereafter the era known as the *second AI winter* started.

Despite the winter, many researchers kept working on neural networks and by 2006 neural networks established the state-of-the-art in many important pattern recognition competitions, gaining popularity again that lasted until today.

Lastly, it should be noted that although neural networks were inspired by the biological neural networks in animal brains, today’s research on neural networks is mostly guided by several quantitative disciplines as mathematics, statistics, computer science and engineering. These disciplines treat neural networks as powerful function

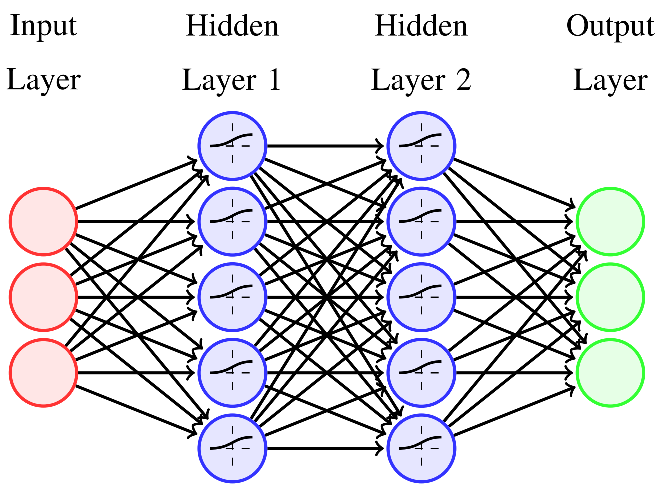


Figure 2.5: An example feed-forward neural network architecture with two hidden layers each having sigmoid activation functions.

approximators and do not care much about their inspiration source.

2.3 Neural Networks

Neural networks (articificial neural networks, multilayer perceptrons) can be considered as perceptrons with multiple layers. The intermediate layers are called the hidden layers and each of the neurons (units) in a hidden layer is connected to all of the neurons in the previous and next layer. The role of these hidden layers is to provide useful inputs to the next layer by combining the information from the previous layers in a hierarchical manner. An example neural network architecture with two hidden layers is given in Figure 2.5.

In a neural network, each layer l applies a linear transformation to the activation vector $\boldsymbol{a}^{(l-1)}$ coming from its previous layer $l-1$. This transformation is done with the layer's parameters $w_{ij}^{(l)}$ that determines the connection strength between the i-th neuron in layer $l-1$ and j-th neuron in layer l. For compactness, these parameters are stored in the parameter matrix $\boldsymbol{W}^{(l)}$. The biases are also stored as $w_{0j}^{(l)}$ inside this matrix in a similar way that was described in Section 2.2. This transformation is given in equation 2.4.

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} \tag{2.4}$$

This linearly transformed input $\boldsymbol{z}^{(l)}$ (pre-activation) is then passed through a non-linear element-wise activation function $f(.)$ to obtain the activation $\boldsymbol{a}^{(l)}$ of layer l as in equation 2.5.

$$\boldsymbol{a}^{(l)} = f(\boldsymbol{z}^{(l)}) \tag{2.5}$$

After performing these subsequent linear transformations followed by non-linearities the overall relation between the input vector $\boldsymbol{x}$ and the output vector $\boldsymbol{y}$ of a neural network with L layers becomes as in in equation 2.6.

$$\boldsymbol{y} = f(\boldsymbol{W}^{(L)} \ldots f(\boldsymbol{W}^{(2)} f(\boldsymbol{W}^{(1)} \boldsymbol{x})) \ldots) \tag{2.6}$$

The number of hidden layers, the number of neurons in a hidden layer and the choice of the activation functions are all called the *hyperparameters*[1] of the neural network and they are often chosen according to performance of the network on a predefined validation set.

Lastly, depending on the direction of information flow, neural networks can be divided into two groups. The neural network in Figure 2.5 is a feed-forward neural network because information flows from its input layer to its output layer. There are no *feedback* connections where the output of the network is fed back to itself. When feed-forward neural networks are extended to have feedback connections, they are called recurrent neural networks. These networks are usually used when the input is a time series. Although these recurrent neural networks can be incorporated in several ways, they are out of the scope of this thesis. Thus the following sections will continue with regular feed-forward neural networks.

[1] Hyperparameters are the parameters that are set prior to the training process.

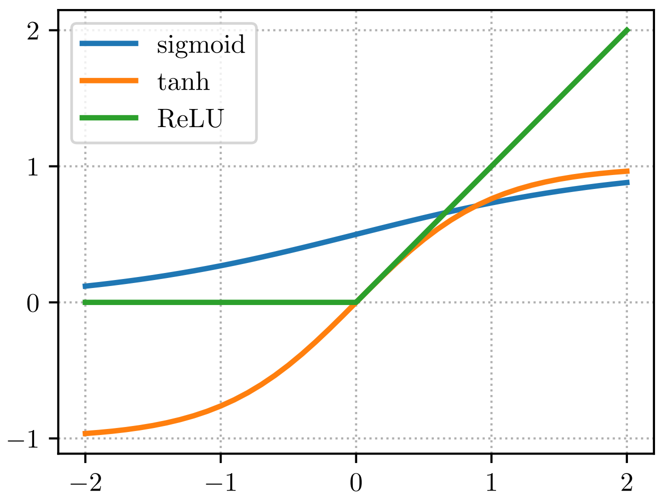


Figure 2.6: Common activation functions used in neural networks.

2.3.1 Activation Functions

Activation functions are non-linear functions that are element-wisely applied to the pre-activation vectors in each layer. These non-linearities map non-linearly separable problems into spaces that are linearly separable. If there were no activation functions, neural networks would just be linearly transforming their inputs and their output would be equivalent to a single linear transformation, i.e., a neural network with one or more hidden layers could have been replaced by a network with no hidden layers at all.[2] As a result, this transformation would have failed in non-linearly separable problems.

The most commonly used activation functions are sigmoids (logistic, σ), hyperbolic tangents (tanh) and rectified linear units (ReLU) and their input-output relationship is depicted in Figure 2.6. The sigmoid function is a squashing function that maps its real-valued inputs into the range between 0 and 1. The mathematical form of the sigmoid and its gradient is given in equation 2.7..

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x)) \tag{2.7}$$

The tanh function is like the sigmoid function, except that it squashes its real-valued

[2] This is a result of multiple steps of linear transformations being equivalent to a single linear transformation.

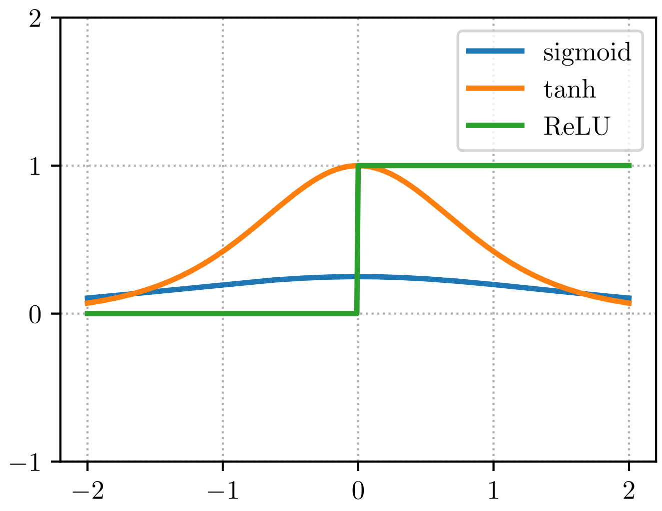


Figure 2.7: Gradients of common activation functions used in neural networks. Notice how the gradients of the sigmoid and the tanh vanish as their input moves away from zero.

inputs into the range between -1 and 1. The mathematical form of tanh and its gradient is given in equations 2.8. It is important to note that the tanh function is just the scaled and translated version of the sigmoid function, i.e., $\tanh(x) = 2\sigma(x) - 1$.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \; \frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x) \tag{2.8}$$

Sigmoid activations used to be the canonical activation functions in neural networks due to their interpretation as the firing rate of biological neurons; however, they have fallen out of favor because of their gradient killing property. When the inputs to the sigmoid activation are far from 0, the local gradient becomes very small during backpropagation and this kills all the gradients flowing to the parameters of itself and its inputs. This usually results in a significant drop in learning speed and it may even cause no learning at all. tanh activations also suffer from the same problem (see Figure 2.7). One advantage of tanh over sigmoid however is its outputs are zero-centered.

Finally, the ReLU activation thresholds the activation at zero resulting in a positive output only if the pre-activation is positive. The mathematical form of the ReLU activation and its gradient is given in equations 2.9.

$$\mathrm{ReLU}(x) = \max(0, x), \ \frac{\partial \mathrm{ReLU}(x)}{\partial x} = \mathrm{step}(x) \tag{2.9}$$

As of today, ReLU is the most widely used activation function in feed-forward neural networks. Its importance in learning was first demonstrated by Jarrett et al. [39] and then by Glorot et al. [40]. It has several advantages than the other activations presented above. First, ReLU is computationally cheap because its output is either zero or the input itself. There is no need to compute time consuming exponentials as in sigmoid and tanh. Second, the linear region of ReLU provides constant gradients of 1 that speeds up the training process. Third, ReLU induces sparsity that allows better representations. One major problem with ReLU however is that too large gradients can cause the weights to change in such a way that will result in the pre-activations of the ReLU neuron to move to the far negative side, resulting in its local gradients to be zero for the entire training period. This is commonly known as the dying ReLU problem and it can be prevented by using small learning rates. Lastly, it should be noted that although ReLU is the popular choice of activation functions nowadays, which activation functions to use with different network architectures and different tasks is still an active area of research.

2.3.2 Neural Networks as Universal Function Approximators

One of the remarkable properties of neural networks is that they can approximate any arbitrary function to a desired degree of accuracy. This is called the universal function approximation property of neural networks. One hidden layer with a non-linear activation function is enough for this property to hold. Although approximating continuous functions with other simpler continuous functions can be dated back to the 1957 study of Kolmogorov [41], this study was not in the context of neural networks. Two of the important studies on this universality theorem in the context of neural networks are the studies from Cybenko [42] and Hornik et al. [43]. For the sake of brevity, rather than providing a concrete mathematical derivation of the theorem, a visual explanation will be provided instead. Figure 2.8 illustrates one way of this

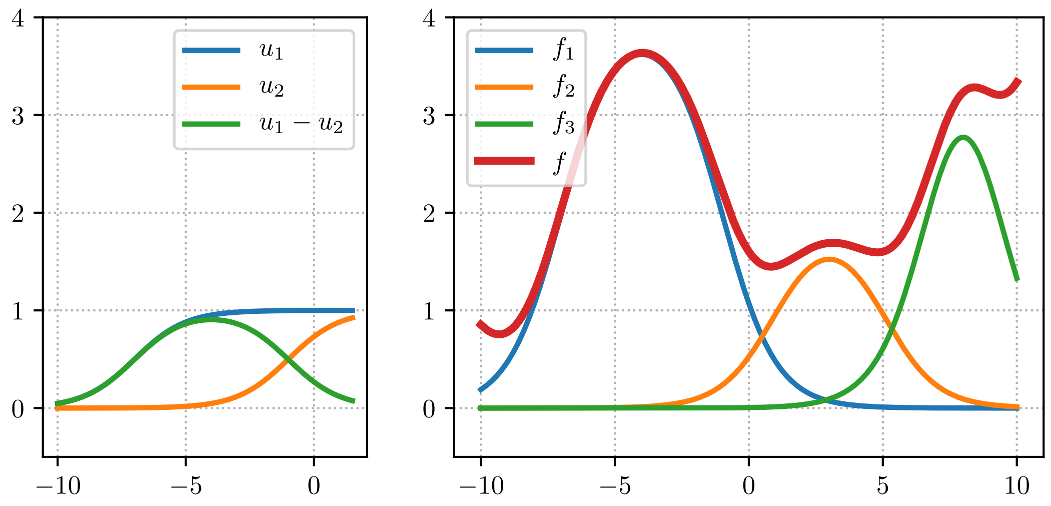


Figure 2.8: Left: Two non-weighted offset sigmoid hidden units forming a local bump function around -4. Right: Three weighted local bump functions f_1, f_2 and f_3 approximating an arbitrary function f.

approximation process for an arbitrary 1D function.

On the left side of Figure 2.8, two non-weighted offset sigmoid hidden units u_1 and u_2 form a local bump function. Other hidden units can also form weighted and translated versions of this bump function as f_1, f_2 and f_3 which are depicted on the right side of Figure 2.8 and then they can all be used together to approximate an arbitrary function f. The more bump functions the neural network has, the more closely it will approximate the desired function.

It should be noted that forming local bumps using squashing functions is just one way of approximating arbitrary functions. Other types of local bump functions can also be formed by hidden units with different activation functions as ReLU. It should be noted that the same approximation idea also applies to higher dimensions.

Lastly, it is important to note that neural networks without non-linearities are not universal function approximators as they can be reduced to simple linear models which consist of just input and output layers. Another thing to note is that although neural networks have the capacity to represent an arbitrary function, it is up to the learning algorithm to learn the function. The approximation theorem guarantees nothing about this and in fact the learning algorithm can fail to represent the desired function due to the common problems as overfitting[3] and underfitting[4].

2.3.3 Deep Networks against Shallow Networks

In Section 2.3.2, it was stated that neural networks with single hidden layers are enough to approximate arbitrary complex functions. This raises questions on the usefulness of using multiple hidden layers. A study by Barron [44] shows that if a single hidden layer is used, the number of hidden units may have to be exponential in the input dimensionality in the worst case which makes the neural network impractically large. Besides making it impractical, having too many hidden units also increases the probability of overfitting as it rapidly increases the number of parameters.

Rather than using shallow networks, using deep networks that have more than a single hidden layer is a better choice for several reasons.[5] First, as shown by Montufar et al. [45] deep networks with ReLU activations can represent functions with a number of regions that is exponential in the depth of that network whereas the number of represented regions is just polynomial with the number of hidden units per layer. This allows deep networks to have less hidden units in each of their layers and thus makes them both practical and less prone to overfitting. Second, rather than just using the activation functions as pieces of a complex function, deep networks can form several functions from simpler ones to approximate a desired function. In fact, this hierarchical representation building is one of the key ideas behind the success of today's deep neural networks.

In summary, deep networks express a useful prior over the space of possible functions that the network can learn and they are more feasible solutions to the problem of function approximation.

[3] The case of learning a model which fits too closely to a particular set of data.
[4] The case of learning a model fails to capture the underlying structure of the data.
[5] There is no agreed certain depth among researchers that separates deep networks from shallow ones.

2.4 Optimization in Neural Networks

This section focuses on gradient-based optimization methods that make use of the gradient of the loss function to optimize the parameters of a neural network. After presenting the commonly used loss functions, an efficient gradient computation method will be discussed. Then, some of the commonly used gradient-based optimization algorithms will be explained. Lastly, the overall picture of neural network training will be presented.

2.4.1 Loss Functions

In neural network training, the data is given and fixed; however, the parameters $\boldsymbol{W}$, where $\boldsymbol{W}$ is the tensor containing all the network parameters, are adjustable and the goal in training is to adjust these parameters in a way to achieve a desired task. Specifically, in supervised learning, the desired task for a neural network is to produce outputs that are consistent with the ground-truth labels of the training/test data. To be able to adjust the parameters, the first step is to define a parametric loss function $\mathcal{L}(\boldsymbol{W})$ (cost function, objective) that acts as a measure of a network's performance. This loss will be high when the network performs in an undesired way and low when it performs in the expected way.

As explained in Section 2.1, two of the most popular supervised learning tasks performed in machine learning are classification and regression. Classification was the task of predicting the class labels of a given example, whereas regression was the task of predicting real-valued labels. The most commonly used loss function for classification tasks is the cross-entropy loss that is given in equation 2.10,

$$\mathcal{L}(\boldsymbol{W}) = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{e^{a_{y_i}}}{\sum_j e^{a_j}}\right) \tag{2.10}$$

where N is the number of training examples or batch size if the dataset is divided into batches, a_{y_i} is the activation in the last layer that corresponds to the label of the i-th example and a_j is the j-th activation in the last layer. For regression tasks on the other hand, the most commonly used loss functions are the L1 loss and the L2 loss (mean squared error, MSE) that are given in equations 2.11 and 2.12 respectively,

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} |y_i - a_i| \tag{2.11}$$

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}(y_i - a_i)^2 \tag{2.12}$$

where N is again the number of training examples or batch size, y_i is the real-valued label of the i-th example and a_i is the activation in last layer that corresponds to the i-th example.

After choosing the loss function, the next step is to choose an optimization algorithm to adjust the parameters of the network in a desired direction. As of today, neural networks are usually trained by gradient-based optimization methods that iteratively adjust the parameters in the reverse direction of the gradient to minimize the loss function. This algorithm is known as gradient descent and it is the core idea behind many state-of-the-art optimization algorithms. However, in order to use these algorithms, the gradient has to be computed first. The next subsection will focus on discussing an efficient way of computing this required gradient.

2.4.2 Gradient Computation in Neural Networks

In this subsection two different approaches to gradient computation will be discussed. The former is a very naive approach that is not used in practice and it will only be used for showing the efficiency of the latter one.

2.4.2.1 The Finite Differences Method

One very naive way of computing the gradient of a loss function $\mathcal{L}(\boldsymbol{W})$ with respect to its parameter $w_{ij}^{(k)}$ is by using the classical finite differences method given in equation 2.13, where ϵ is a very small positive number that is only added to a single parameter whose gradient is to be computed.

$$\frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial w_{ij}^{(k)}} = \lim_{\epsilon \to 0} \frac{\mathcal{L}(w_{ij}^{(k)} + \epsilon) - \mathcal{L}(w_{ij}^{(k)})}{\epsilon} \tag{2.13}$$

This method is conceptually simple as it requires just two forward passes to compute the gradient of a single parameter. However, this operation has to be done for every single parameter $w_{ij}^{(k)}$. Considering the fact that today's neural networks have tens of millions of parameters, this method can be very slow and it is thus not used in practice. Another downside of this method is that it computes the approximate gradient rather than the exact one.

2.4.2.2 Backpropagation

Backpropagation (reverse-mode automatic differentiation) is an efficient method for calculating the gradients using the chain rule from calculus. It propagates the gradient information of the loss function from the last layers of the network to the initial layers. Unlike the finite differences method, backpropagation calculates the gradient in a single pass rather than requiring two passes for every single parameter. In today's large networks, this can make it millions of times faster than the naive finite differences method.[6] Additionally, backpropagation calculates the exact gradient instead of the approximate one.

Understanding backpropagation with an example is much easier. Consider the neural network architecture given in Figure 2.5, it takes a three dimensional input $\boldsymbol{x}$ and after passing it through two hidden layers outputs a three dimensional output $\hat{\boldsymbol{y}}$. In order to distinguish it from the desired output $\boldsymbol{y}$, the neural network output is denoted as $\hat{\boldsymbol{y}}$. Also consider a regression task with an L2 loss as in equation 2.14,

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N} \sum_{\mathcal{D}} \frac{1}{2} (\boldsymbol{y} - \hat{\boldsymbol{y}})^2 \tag{2.14}$$

where the summation is over the whole dataset $\mathcal{D}$ with N examples and $\hat{\boldsymbol{y}}$ is a function

[6] This is the difference between a neural network taking a day to train and taking at least $\sim$ 2700 years.

of the input $\boldsymbol{x}$ and parameters $\boldsymbol{W}$. Then, the derivative of the loss with respect to each of the parameters $w_{ij}^{(l)}$ of the network is as in equation 2.15,

$$
\begin{aligned}
\frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial w_{ij}^{(l)}} &= \frac{\partial}{\partial w_{ij}^{(l)}} \left(\frac{1}{N} \sum_{\mathcal{D}} \frac{1}{2} (\boldsymbol{y} - \hat{\boldsymbol{y}})^2 \right) \\
&= \frac{1}{N} \sum_{\mathcal{D}} \left(\frac{1}{2} \frac{\partial}{\partial w_{ij}^{(l)}} \left((\boldsymbol{y} - \hat{\boldsymbol{y}})^T (\boldsymbol{y} - \hat{\boldsymbol{y}}) \right) \right) \\
&= \frac{1}{N} \sum_{\mathcal{D}} \left(-(\boldsymbol{y} - \hat{\boldsymbol{y}})^T \frac{\partial(\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} \right)
\end{aligned} \tag{2.15}
$$

where backpropagation allows the computation of $\frac{\partial(\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}}$ by exploiting the chain rule. Considering, e.g., that the derivative of the loss with respect to one weights in the first layer $w_{ij}^{(1)}$ is to be computed, the expansion will continue as in equation 2.16,

$$
\begin{aligned}
\frac{\partial \hat{\boldsymbol{y}}}{\partial w_{ij}^{(1)}} &= \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(3)}} \cdot \frac{\partial \boldsymbol{z}^{(3)}}{\partial w_{ij}^{(1)}} \\
&= \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(3)}} \cdot \frac{\partial \boldsymbol{z}^{(3)}}{\partial \boldsymbol{a}^{(2)}} \cdot \frac{\partial \boldsymbol{a}^{(2)}}{\partial w_{ij}^{(1)}} \\
&= \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(3)}} \cdot \frac{\partial \boldsymbol{z}^{(3)}}{\partial \boldsymbol{a}^{(2)}} \cdot \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} \cdot \frac{\partial \boldsymbol{z}^{(2)}}{\partial w_{ij}^{(1)}} \\
&= \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(3)}} \cdot \frac{\partial \boldsymbol{z}^{(3)}}{\partial \boldsymbol{a}^{(2)}} \cdot \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} \cdot \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{a}^{(1)}} \cdot \frac{\partial \boldsymbol{a}^{(1)}}{\partial w_{ij}^{(1)}} \\
&= \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(3)}} \cdot \frac{\partial \boldsymbol{z}^{(3)}}{\partial \boldsymbol{a}^{(2)}} \cdot \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} \cdot \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{a}^{(1)}} \cdot \frac{\partial \boldsymbol{a}^{(1)}}{\partial \boldsymbol{z}^{(1)}} \cdot \frac{\partial \boldsymbol{z}^{(1)}}{\partial w_{ij}^{(1)}}
\end{aligned} \tag{2.16}
$$

where the first five terms in the last expression are Jacobian matrices[7] and the last one is the derivative of the pre-activation $\boldsymbol{z}^{(1)}$ with respect to the weight $w_{ij}^{(1)}$. Replacing $\hat{\boldsymbol{y}}$ by the corresponding activation $\boldsymbol{a}^{(3)}$ and using equations 2.4 and 2.5, the last expression in equation 2.16 becomes the expression in equation 2.17,

$$
\frac{\partial \hat{\boldsymbol{y}}}{\partial w_{ij}^{(1)}} = \frac{\partial \boldsymbol{a}^{(3)}}{\partial \boldsymbol{z}^{(3)}} \cdot \boldsymbol{W}^{(3)} \cdot \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} \cdot \boldsymbol{W}^{(2)} \cdot \frac{\partial \boldsymbol{a}^{(1)}}{\partial \boldsymbol{z}^{(1)}} \cdot [x_j]_i \tag{2.17}
$$

where $[x_j]_i$ is a vector with the same dimensionality of $\boldsymbol{z}^{(1)}$ whose i-th element is

[7] The matrix representing the derivative of a vector-valued function with respect to a vector-valued input.

equal to the j-th element of $\boldsymbol{x}$ and all the other elements are zero. It is also important to note that the input $\boldsymbol{x}$ can be treated as the activation of the 0-th layer, i.e., $\boldsymbol{x} = \boldsymbol{a}^{(0)}$. In this case, $[x_j]_i$ can be replaced by $[a_j^{(0)}]_i$.

Each derivative $\frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}$ is an $n \times n$ matrix as in equation 2.18,

$$\frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} = \begin{bmatrix} \dfrac{\partial a_0^{(l)}}{\partial z_0^{(l)}} & \cdots & \dfrac{\partial a_0^{(l)}}{\partial z_n^{(l)}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial a_n^{(l)}}{\partial z_0^{(l)}} & \cdots & \dfrac{\partial a_n^{(l)}}{\partial z_n^{(l)}} \end{bmatrix} \tag{2.18}$$

where n is the dimensionality of $\boldsymbol{a}^{(l)}$ and $\boldsymbol{z}^{(l)}$. This matrix is also known as the Jacobian of $\boldsymbol{a}^{(l)}$ with respect to $\boldsymbol{z}^{(l)}$. Because of the element-wise non-linearities that map $z_i^{(l)}$ to $a_i^{(l)}$ as $a_i^{(l)} = f(z_i^{(l)})$, the Jacobian of the activations with respect to the pre-activations is a diagonal and sparse matrix as in equation 2.19,

$$\frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} = \begin{bmatrix} \dfrac{\partial a_0^{(l)}}{\partial z_0^{(l)}} & & \Large 0 \\ & \ddots & \\ \Large 0 & & \dfrac{\partial a_n^{(l)}}{\partial z_n^{(l)}} \end{bmatrix} \tag{2.19}$$

This sparsity can be exploited to speed up the computations and in fact it is done by modern frameworks. It should be noted that the backpropagation algorithm developed above is just the pure algebraic version. Modern frameworks perform these algebraic computations on graphs (to model dependency and to build a modular approach) and they use dynamic programming techniques that store intermediate values of computations along the way for gaining efficiency. The implementation details of these frameworks are out of the scope of this thesis. Interested readers can find more information in Chapter 6 (Section 6.5) of the Deep Learning book of Goodfellow et al. [26].

As of today, there are many open-source automatic differentiation frameworks released by major companies and research labs. These frameworks exploit GPUs to perform high-performance computations. In this thesis, one of these frameworks, released by Facebook AI Research (FAIR), called PyTorch [46] is used.

After the gradient computation, the gradient-based optimization algorithms will use it to perform updates to the parameters towards a specific goal. The details of these algorithms will be presented in the following subsection.

The History Behind Backpropagation

The continuous form of backpropagation was developed at least in 1960 by Kelley [47] and in 1961 by Bryson [48] using the Euler-Lagrange equations and the chain rule. In 1962, Dreyfus published a simpler derivation of backpropagation using only the chain rule [49]. See also the the other studies in the 1960s: Bryson and Denham 1961 [50], Pontryagin 1961 [51], Wilkinson 1965 [52], Amari 1967 [53], Bryson and Ho 1969 [54], Director and Rohrer 1969 [55]. Although these methods used dynamic programming [56] for efficiency, they backpropagated the error gradients by passing the Jacobian matrix from one layer to the previous layer and did not explicitly take into account the possible efficiency gains due to the sparsity of these Jacobians. They were also developed in the context of control theory.

Efficient backpropagation, which addressed the sparsity of Jacobians, was first described and implemented in Linnainmaa's 1970 master's thesis [57, 58]. However, Linnainmaa used it for estimating the effects of arithmetic rounding errors on the results of complex expressions rather than using it for gradient computation in neural networks. After Dreyfus used this efficient backpropagation to minimize parametric cost functions in 1973 [59], Werbos was the first to consider to possibility of applying it neural networks in his 1974 PhD thesis [60] and in 1981 he was also the first to apply efficient backpropagation to neural networks as it is used today [61]. However, it was the 1986 paper of Rumelhart et al. [38] that significantly contributed to the popularization of backpropagation in neural networks by demonstrating that it can generate useful representations inside the hidden layers of the network.

The core ideas behind the backpropagation algorithm used in the 1981 study of Werbos and in the 1986 study of Rumelhart et al. are still used today for gradient computations in neural networks. The performance increase in today's networks however is mostly due to the larger datasets, more powerful computers and some algorithmic improvements. Interested readers can refer to the deep learning review paper of Schmidhuber [62] for a more thorough review of the studies that contributed to today's state-of-the-art neural networks.

2.4.3 Gradient-Based Optimization Algorithms

In this subsection, four different approaches to updating the parameters of neural networks will be discussed. All of these approaches actually perform gradient descent, however, they also use several tricks to speed up the optimization process.

2.4.3.1 Stochastic Gradient Descent

Gradient descent (GD) [63] is a popular optimization algorithm for optimizing neural networks. In GD, after the gradient computation, a small step in the reverse direction of the gradient is taken to minimize a loss function $\mathcal{L}(\boldsymbol{W})$. After repeating this operation for a finite number of steps the parameters will converge to a local or global minimum. The parameter update equation is as in equation 2.20,

$$\boldsymbol{W} := \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W}) \tag{2.20}$$

where α is called the learning rate. This learning rate is used in scaling the gradient so that it does not perform a huge update.

It should be noted that computing the loss function using the whole dataset can take some time. Because of this, the loss function is usually computed using batches from the dataset. When batches of the data are used, GD takes the prefix "stochastic" and becomes stochastic (or batch) gradient descent (SGD). SGD has the same update equation as in equation 2.20; however, the $\mathcal{L}(\boldsymbol{W})$ term is calculated using batches of data. Because of this, $\nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})$ gives the approximate gradient. However, in practice, this does not cause serious issues because the learning rate makes the update step very small and taking a small step in a slightly wrong direction does not cause much of an issue.

If the loss function $\mathcal{L}(\boldsymbol{W})$ is convex, as in the case of linear models, SGD converges to a global minimum. However, when neural networks or any other non-linear models are used, $\mathcal{L}(\boldsymbol{W})$ becomes non-convex and there is no guarantee that SGD will converge to a global minimum. In fact, converging to a local minimum is much more probable.

2.4.3.2 Stochastic Gradient Descent with Momentum

Vanilla SGD can be very slow to converge. Thus, in practice, SGD is often used with momentum [38]. Momentum is a physical perspective to optimization which is inspired from the momentum concept in physics. The idea is to build up a velocity in places where the gradient is large and use this to rapidly update the model parameters. This allows SGD to converge much faster. Another advantage of using momentum is to stabilize the SGD updates at minimums. The two-step parameter update equation is as in equation 2.21,

$$\begin{aligned} \boldsymbol{v} &:= \mu\boldsymbol{v} - \alpha\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W}) \\ \boldsymbol{W} &:= \boldsymbol{W} + \boldsymbol{v} \end{aligned} \tag{2.21}$$

where α is the learning rate, μ is the momentum and $\boldsymbol{v}$ is the velocity. In practice, $\boldsymbol{v}$ is initialized at zero and μ is set to 0.9 for damping purposes.

2.4.3.3 RMSProp

Besides its slowness, vanilla SGD also suffers from using the same learning rate for every parameter of the network. This can be a problem when the learning rate has to be tuned carefully. RMSProp [64] eliminates the need for manually tuning the learning rate by keeping a running average of the previous gradients and scaling the learning rate using this average. By this way, the learning rates of the parameters with large recent gradient history will become smaller, preventing the huge updates. The two-step parameter update equation of RMSProp is as in equation 2.22,

$$\begin{aligned} \boldsymbol{g} &:= \gamma \boldsymbol{g} + (1-\gamma)[\nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})]^2 \\ \boldsymbol{W} &:= \boldsymbol{W} - \frac{\alpha}{\sqrt{\boldsymbol{g}} + \epsilon} \nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W}) \end{aligned} \tag{2.22}$$

where α is the learning rate, γ is the decay factor that determines the importance of previous gradients in the running average and ϵ is a small constant that prevents division by zero errors. In practice, $\boldsymbol{g}$ is initialized at zero, γ and ϵ are set to 0.9 and 10^{-8} respectively and values in the range $[10^{-4}, 10^{-3}]$ are used for α.

2.4.3.4 Adam Optimizer

Adaptive Moment Estimation (Adam) [65] is an optimization method that combines the powers of momentum and RMSProp. Different from RMSProp, Adam uses the averaged gradient $\boldsymbol{m}$ instead of the raw gradient $\nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})$. Using the average gradient provides stability when the gradients are too noisy. Adam is one of the default optimization methods in today's neural network optimization and its three-step parameter update equation is as in equation 2.23,

$$\begin{aligned} \boldsymbol{m} &:= \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W}) \\ \boldsymbol{g} &:= \beta_2 \boldsymbol{g} + (1-\beta_2)[\nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})]^2 \\ \boldsymbol{W} &:= \boldsymbol{W} - \frac{\alpha}{\sqrt{\boldsymbol{g}} + \epsilon} \boldsymbol{m} \end{aligned} \tag{2.23}$$

where α is the learning rate, ϵ is a small constant preventing zero division errors and β_1 and β_2 are decay factors that determine the importance of the previous gradients in the running averages. In practice, $\boldsymbol{m}$ and $\boldsymbol{g}$ are initialized at zero, β_1, β_2 and ϵ are set to are set to 0.9, 0.999 and 10^{-8} respectively and values in the range $[10^{-4}, 10^{-3}]$ are used for α.

The full Adam update also takes into account the bias correction mechanism which compensates for the zero initialization of $\boldsymbol{m}$ and $\boldsymbol{g}$ when β_1 and β_2 are close to 1, i.e., $\boldsymbol{m}$ and $\boldsymbol{g}$ tend to stay around zero when they are initialized as zero. When bias correction is applied, the parameter update equation turns into a five-step update and it is as in equation 2.24 where t denotes the current iteration number. $\boldsymbol{m}_t$ and $\boldsymbol{g}_t$ are a function of t and as t increases the parameter update becomes the same update as in equation 2.23.

$$\begin{aligned} \boldsymbol{m} &:= \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W}), \quad \boldsymbol{m}_t := \frac{\boldsymbol{m}}{1-\beta_1^t} \\ \boldsymbol{g} &:= \beta_2 \boldsymbol{g} + (1-\beta_2)[\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W})]^2, \quad \boldsymbol{g}_t := \frac{\boldsymbol{g}}{1-\beta_2^t} \\ \boldsymbol{W} &:= \boldsymbol{W} - \frac{\alpha}{\sqrt{\boldsymbol{g}_t}+\epsilon}\boldsymbol{m}_t \end{aligned} \tag{2.24}$$

2.4.3.5 Choosing the Right Optimization Algorithm

Although methods as RMSProp and Adam, often called adaptive methods, seem to be using much more hyperparameters than vanilla SGD and momentum SGD, these hyperparameters are generally fixed and they provide robustness to different learning rates. Adaptive methods also perform better when the input data is sparse as they perform larger updates to rarely occurring features and smaller updates to the frequently occurring ones. They also converge much faster. A study of Schaul et al. [66] compares the above methods together with the other state-of-the-art ones and shows that there is no single best algorithm for performing optimization.

2.4.4 Training Neural Networks - The Overall Picture

In this section, the overall picture of training neural networks using gradient-based methods will be described. Before going any further it is important to remember that the goal in training is to learn a function that maps the training/test data to the training/test labels. Considering that there is a big dataset consisting of training vector pairs $(\boldsymbol{x}, \boldsymbol{y})$, the first step is to divide the dataset into batches so that it can fit in memory and the gradient computation in the further steps does not take much time. After the batch division, a single data batch $\boldsymbol{X}_i$ is passed forward through the untrained network to obtain the output batch $\hat{\boldsymbol{Y}}_i$ which is both a function of the input $\boldsymbol{X}_i$ and the network parameters $\boldsymbol{W}$. This output batch $\hat{\boldsymbol{Y}}_i$ together with the label batch $\boldsymbol{Y}_i$ are then used in computing the loss function $\mathcal{L}(\boldsymbol{W})$. Since the parameters are initially random, the loss will be very high at first. With the computation of the loss, the *forward pass* is completed and the next step is to assign credits to the parameters who caused the high loss which is referred to as the *backward pass*.

The backward pass starts with computing the gradient of the loss $\mathcal{L}(\boldsymbol{W})$ with respect to the network parameters $\boldsymbol{W}$. This is done by the backpropagation algorithm in a backwards manner, i.e., it first computes the gradient with respect to the parameters of the last layer $\boldsymbol{W}^{(L)}$, then with respect to $\boldsymbol{W}^{(L-1)}$ and so on. After the gradient computation, the gradient-based optimization algorithm uses it to slightly update the parameters in the reverse direction of the gradient which in turn minimizes the loss a bit. With this update the backward pass is completed.

In order for the network to observe the whole dataset, the forward and backward passes have to be done for every batch pair $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$. After performing these forward and backwards passes over the whole dataset for finite number of times, the neural network's parameters $\boldsymbol{W}$ will converge to nice values that map the training data to the training labels with a small amount of error. This whole process is referred to as *training a neural network*.

2.5 Convolutional Neural Networks

Section 2.3 introduced vanilla neural networks that performed linear transformations followed non-linear mappings in each of its layers. In these networks, all of the neurons in a layer is connected to all of the neurons in its previous and next layer. Because of this connectivity, these networks are often referred to as *fully-connected* neural networks. When the input data has a structure, this fully-connectedness might not be necessary and a connectivity pattern that exploits this structure can be a more clever choice. An advantage of this choice is to reduce the number of parameters which will in turn reduce both the memory usage and the computation time. Reducing the number of parameters will also prevent some of the possible problems as overfitting, forcing the network to focus on the important points in the input.

Convolutional neural networks (CNN, ConvNet) are example architectures that exploit the structure of the input data. The assumption is that input data as images and

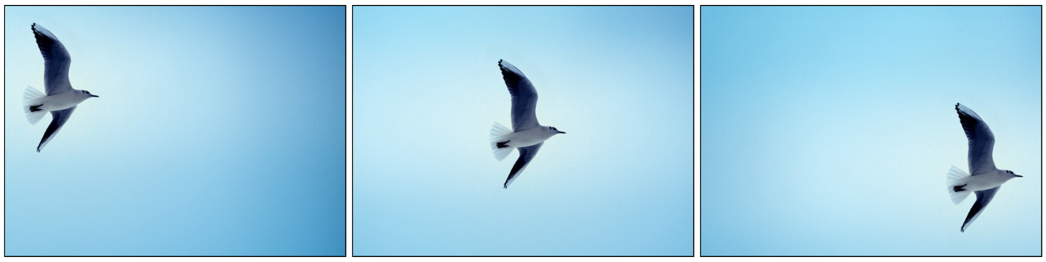


Figure 2.9: The same bird in three different parts of an image.

audio have a local topological structure (locality) that does not depend on its location (translational invariance). In the case of images, this means that a bird can appear in different parts of the image as in Figure 2.9 and it is the same bird in all three of them.

CNNs exploit this structure by applying the same parametric pattern detectors in different parts of the image.[8] This is formally done by an operation called convolution, the operation that gives the name to the neural network. It basically superimposes and then multiplies the image with a parametric pattern detector (kernel, filter) to produce an activation at each point. At the end of the convolution operation, a matrix of activations, which is often referred to as feature map, is obtained. In practice, multiple kernels are applied with the same convolution operation and this results in an output tensor containing multiple feature maps.

The convolution operation is just a repetition of two operations: superimpose-multiply and shift (stride). When the stride is 1 pixel, the repetition of these two operations work without any problems. However, if the stride is greater than 1, in some cases some parts of the kernel can superimpose with the regions outside the borders of the image and this can cause a problem. The typical solution is to add a frame of zeros around the image, often referred to as padding.

The elements described above define the convolution operation and they form the convolutional layer. CNNs also have pooling layers and fully-connected layers whose details will be formally discussed in the following subsections. An example CNN architecture is given in Figure 2.10.

[8] This property of using the same pattern detector is often referred to as weight sharing.

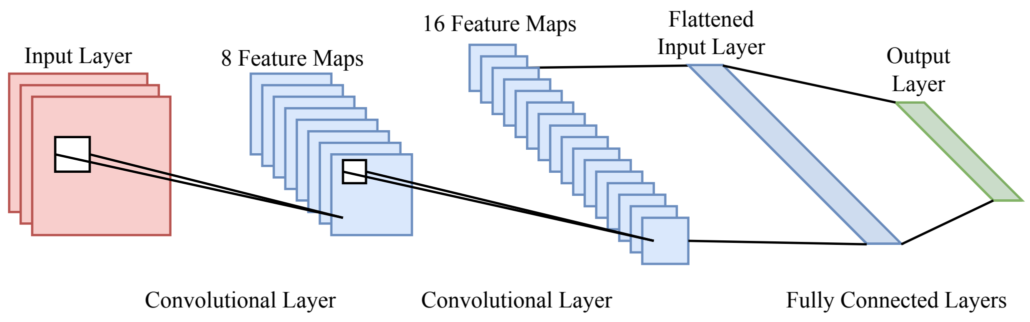


Figure 2.10: An example CNN architecture with two convolutional layers followed by a fully-connected layer.

2.5.1 Convolutional Layer

The convolutional layers of CNNs perform convolution operations on the input feature maps using its parametric kernels and then passes its output through an element-wise non-linearity to obtain the feature maps for the next layer. Figure 2.11 provides an example 2D convolution operation.[9] To keep the drawing simple, a single input feature map is provided; however in practice, the input is usually a tensor where multiple feature maps are stacked together.

In Figure 2.11, a 3×3 kernel[10] slides across the blue input feature map and at each location, the product of the kernel with the area it superimposes (receptive field) is taken to obtain the green pre-activation map. This procedure can be repeated with different kernels to obtain as many pre-feature maps as desired. If the input was a stacked feature map, then the kernel would also have to be a stacked kernel where each kernel would have convolved each map and the resulting pre-feature maps would have summed up to obtain the overall pre-feature map in the end.

To be more formal and general, consider the following parameters:

- i_j: input size along the j-th axis
- o_j: output size along the j-th axis

[9] This figure and the following ones on convolution arithmetic were drawn by using the publicly available code of Dumoulin and Visin's Convolution Arithmetic Guide [67].

[10] Whose weights are the small numbers in the bottom right parts of the shaded blue activations.

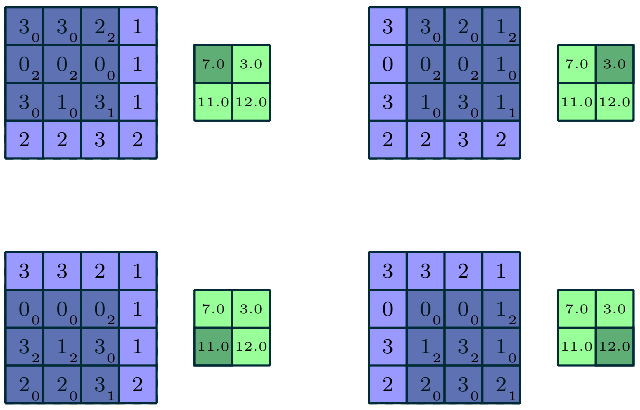


Figure 2.11: An example convolution operation where a 3×3 kernel slides across a 4×4 input feature map to produce a 2×2 pre-feature map. There is no zero-padding and the stride is 1.

- k_j: kernel size along the j-th axis
- s_j: stride along the j-th axis
- p_j: zero-padding along the j-th axis
- N: dimensionality of the convolution operation

where the first one is a property of the input, second one of the output and last four of the convolution operation. The relationship between these parameters is as in equation 2.25,

$$o_j = \frac{i_j - k_j + 2p_j}{s_j} + 1, \; j = 1, \ldots, N \tag{2.25}$$

which tells that the output size o_j increases with p_j and decreases with k_j and s_j. Another important relationship to keep in mind is the number of kernels is equal to the number of output pre-feature maps. After being passed through a non-linearity, these pre-feature maps will all be concatenated to form an additional axis and these concatenated maps will be used as an input to the next layer.

For the sake of simplicity in explaining convolutions, the following parameter assumptions will be made:

- 2D convolution operations ($N = 2$)
- square inputs ($i_1 = i_2 = i$)
- square kernels ($k_1 = k_2 = k$)
- same strides along axes ($s_1 = s_2 = s$)
- same zero-padding along axes ($p_1 = p_2 = p$)

though it should be kept in mind that the same ideas also apply to all other cases where the convolutions are not 2D and the parameters along axes are not equal. It should also be noted that these assumptions would also imply $o_1 = o_2 = o$ from equation 2.25. In Figure 2.11, these assumptions were made and it illustrates an example convolution operation where $i = 4$, $k = 3$, $s = 1, p = 0$. Another example convolution operation with the parameters $i = 3$, $k = 3$, $s = 1, p = 1$ is illustrated in Figure 2.12. In this Figure, as suggested by equation 2.25, the padding preserves the input's dimensionality.

To understand the effects of the stride s and the zero-padding p on the convolution mechanics, consider the following two cases where $i = 5$ and $k = 3$:

- First, consider the cases where the zero-padding p is kept constant, e.g., $p = 0$, and the strides are $s = 1$ and $s = 2$ as illustrated in Figure 2.13. By the time the $s = 1$ convolution sweeps the top of the input, the $s = 2$ one sweeps the whole input. However, this speed increase comes with an information loss penalty. As the stride gets bigger, the pre-feature map gets smaller which in turn results in less information to be passed to the next layers. However, in practice, using a stride that is greater than 1, especially in the first convolutional layers of the network, is very common as the speed-up is substantial compared to the information loss.

- Next, consider the cases where the stride s is kept constant, e.g., $s = 1$, and the zero-paddings are $p = 0$ and $p = 1$ as illustrated in Figure 2.14. While the output size gets decreased in the $p = 0$ case, it stays the same in the other one. It should be noted that when $s = 1$, a padding of $(k - 1)/2$ preserves the resolution of the input.

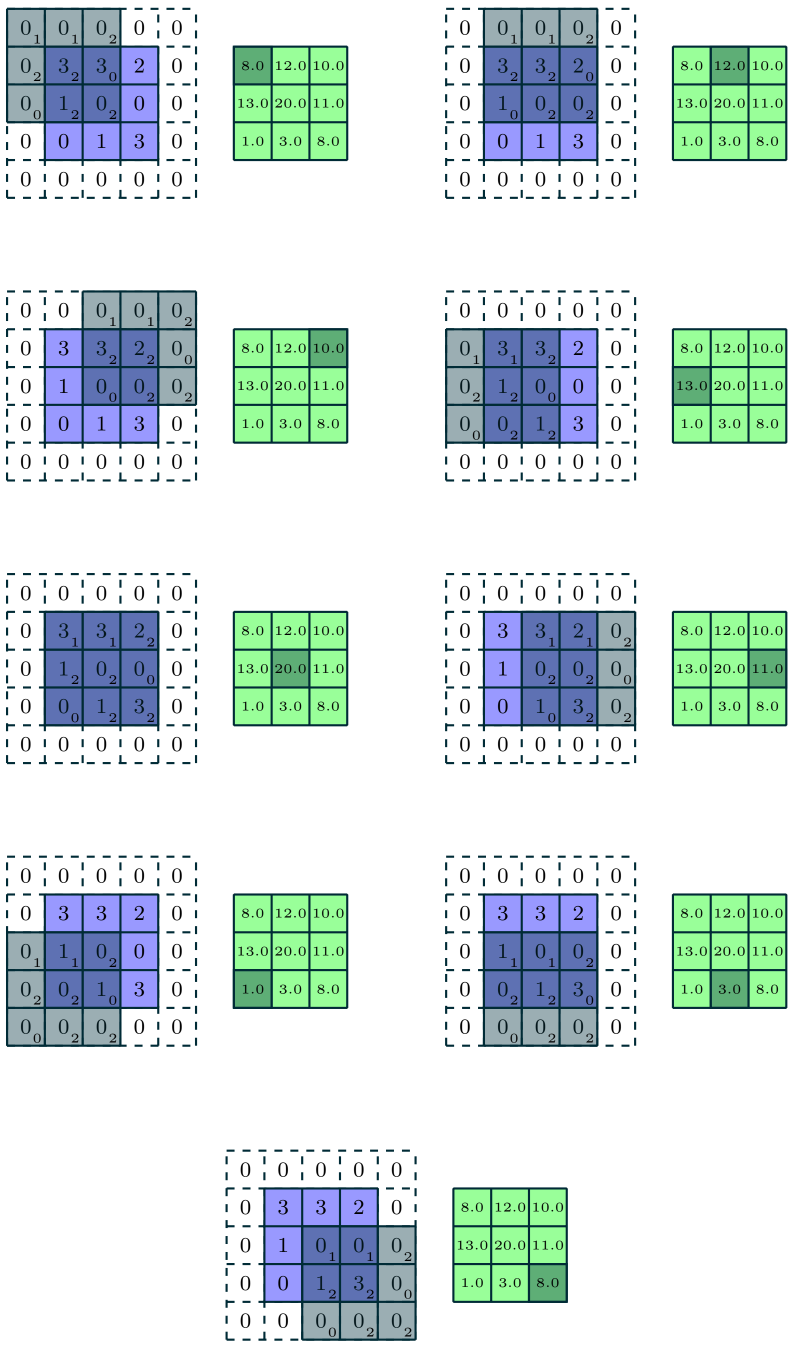


Figure 2.12: An example convolution operation where a 3×3 kernel slides across a zero-padded 3×3 input activation map to produce a 3×3 output pre-feature map. The zero-padding preserves the input's dimensionality. If no padding was used, the output would be 1×1.

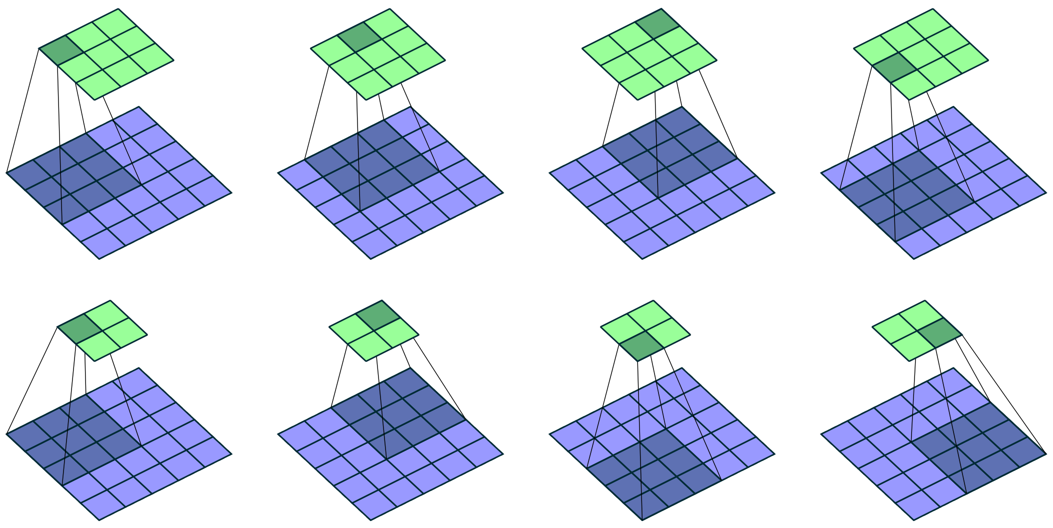


Figure 2.13: The effect of stride on the computation speed. Top Row: The kernel moves one step at a time, i.e., $s = 1$. Bottom Row: The kernel moves two steps a time, i.e., $s = 2$.

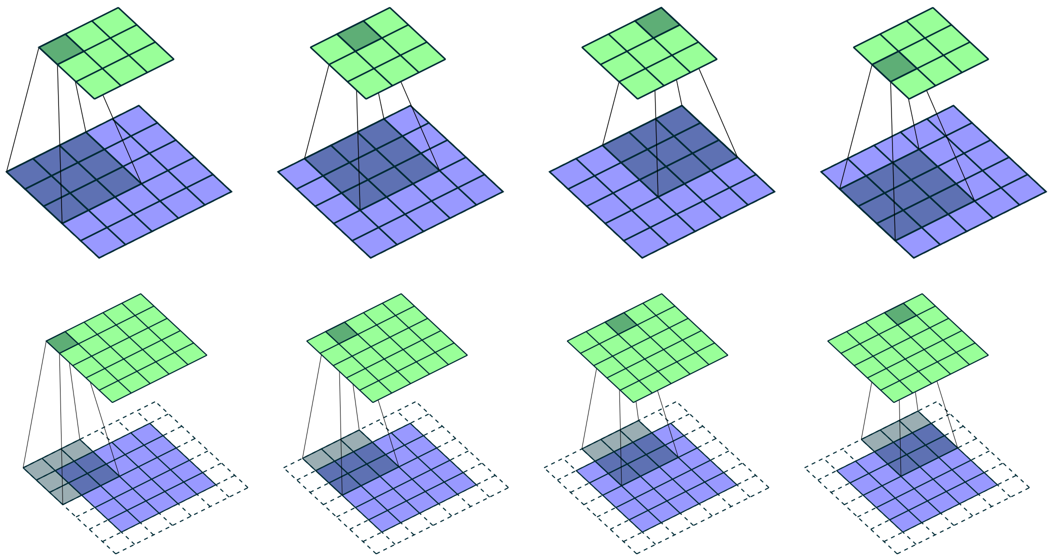


Figure 2.14: The effect of zero-padding on the output size. Top Row: The kernel moves on the $p = 0$ padded input. Bottom Row: The kernel moves on the $p = 1$ padded input.

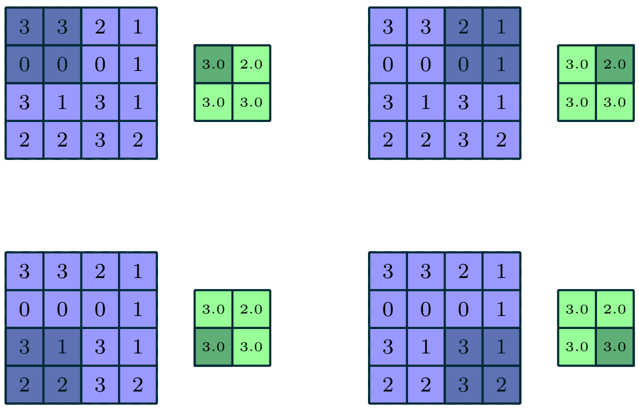


Figure 2.15: An example max pooling operation where a 2×2 window slides across a 4×4 input feature map with stride 2 to produce the 2×2 output feature map.

Lastly, the convolution operation can also be viewed as a local linear transformation where a parametric kernel linearly transforms the local regions of the input. These transformed inputs are then passed through an activation function. In this sense, the convolutional layers of CNNs do the same thing with the hidden layers of vanilla neural networks, except that they do it in a local manner. As discussed above, this locality allows them to exploit the structure present in the input data.

2.5.2 Pooling Layer

The pooling layer is another important building block of CNNs that allows invariance to small translations in the input. It performs pooling operations that reduce the size of the input feature maps and thus allows faster processing. This is done by using functions that summarize subregions of the feature maps. Two of the most popular pooling operations are average pooling and max pooling. Figure 2.15 provides an example of a 2D max pooling operation. Again, for simplicity of the drawings, a case where a single feature map is used as input is provided.

In Figure 2.15, a 2×2 window slides across the input feature map and at each location it takes the max of the region that it superimposes with. At the end of this process, the green feature map is obtained. As in the convolutional layer case, this operation can also be done for the stacked feature map case by doing the same operation for every single feature map in the stack.

Using the parameters defined in the convolutional layer section, the relationship between the size of the input and output of a pooling operation is as in equation 2.26:

$$o_j = \frac{i_j - k_j}{s_j} + 1,\ j = 1, \ldots, N \tag{2.26}$$

which again tells that the output size o_j decreases with both k_j and s_j. It should also be noted that there is no padding operation in pooling.

2.5.3 Fully-Connected Layer

After the input passes through the convolutional and pooling layers, it is first flattened to become a vector and then it is fed as input to the fully-connected layers which are just layers of a regular vanilla neural network. These layers map the extracted features to the desired outputs of the network. It should be noted that since there is no weight sharing in vanilla neural networks, the fully-connected layers contain most of the parameters of a CNN. Because of this, in practice, using too many neurons in these layers is usually avoided.

The History Behind Convolutional Neural Networks

The history behind CNNs can be dated back to the 1960s studies of Hubel and Wiesel [68, 69] in which they studied a cat's early visual cortex. In their studies, they accidentally found that a group of simple and complex cells in their visual cortex fire in response to certain properties of the visual input as edge orientations. The complex cells were also found to have more translational invariance compared to the simple cells. These neurophysiological discoveries were incorporated in Fukushima's Neocognitron [70, 71], the first CNN, in 1979-1980. The Neocognitron consisted of convolutional and subsampling layers that modeled the simple and complex cells of a cat's visual cortex respectively. However, importantly, the weights of the Neocognitron were not learned by backpropagation, but rather set by hand or by unsupervised learning methods [72].

In 1989, backpropagation was applied to Neocognitron-like structures for the first time by Lecun et al. [73]. These CNNs which are trained with backpropagation, augmented with max-pooling layers and accelerated by GPUs form the basis of today's modern state-of-the-art CNNs. Despite their success in the 1980s, the current popularity of CNNs is mostly due to their success in the computer vision community where a deep CNN, called AlexNet [74], was used to win the 2012 ImageNet challenge [75].

2.6 Practical Tricks for Training Neural Networks

This section discusses some the commonly used practical tricks that make neural network training better. After discussing some data preprocessing and weight initialization techniques, we will move on to some regularization methods and batch normalization. It should be noted that none of these methods are mandatory in the training process; however, they can bring a significant performance increase when they are used.

2.6.1 Data Preprocessing

Before feeding the data to the network it is a common practice to preprocess it for standardization. The most common form of preprocessing is mean subtraction where the dataset mean is subtracted across every individual feature of the dataset. It has the geometric interpretation of centering the cloud of data around the origin in every dimension. (see Figure 2.16) Another common preprocessing method is normalization where the data is normalized across every dimension so that they are approximately at the same scale. This normalization is usually achieved by dividing in each dimension to its standard deviation once it has been zero-centered. While working with image inputs specifically, channel mean subtraction is very common. However, normalization is not necessary as the relative scales of RGB values (0-255) are already equal.

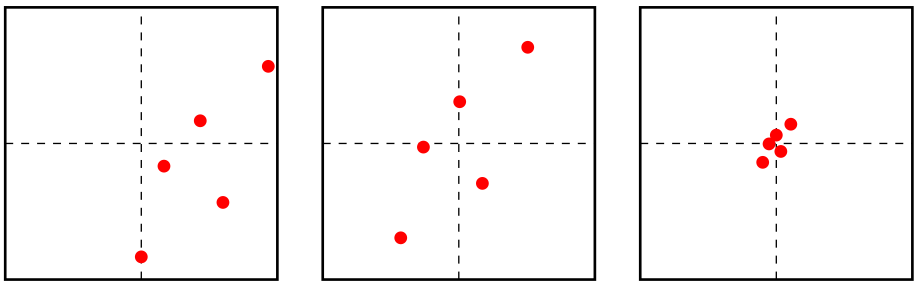


Figure 2.16: Left: Data before preprocessing. Middle: Data after mean subtraction. Right: Data after normalization.

2.6.2 Weight Initialization

Before the training process, the weights and biases of a neural network have to be initialized. This initialization is very important as poor initialization can cause serious problems in the learning process. One very naive way of initializing the weights is by setting all of them to zero. If this is the case, then each of these weights will get the same gradient during backpropagation and there will be no source of asymmetry which is very bad for the training process. A better way would be to initialize the weights from a scaled uniform distribution so that they will be updated in different directions. However, this method also has its own problems as a neuron's output will have a variance that grows with its number of inputs. It turns out that this variance can be normalized by scaling its weight vector by its number of inputs (fan-in, n). This variance normalization has empirical benefits and it is employed in common initializers as the He Initializer [76] where the weights are sampled from a standard normal distribution are scaled with $\sqrt{\frac{2}{n}}$. Biases on the other hand, are usually set to zero as the weights already provide the necessary asymmetry breaking.

2.6.3 Regularization

Regularization methods are useful techniques that prevent overfitting in neural networks. Although there are many types of regularization methods, in this subsection, only Dropout [77] and the L2 regularization will be explained as they are the only ones that are used in this thesis.

Dropout is an important regularization technique that adds useful noise to the hidden units of a neural network. This noise is added by multiplying the hidden units with Bernoulli distributed random variables which take the value 1 with probability p and 0 with probability $1 - p$. Importantly, its operation in the training and evaluation modes are quite different. During training time, the information flows through the noisy network where the features x_k from the feature vector $\boldsymbol{x} = (x_0, \ldots, x_n)$ gets multiplied by independent Bernoulli random variables a_k as in equation 2.27,

$$\hat{x}_k = a_k \frac{1}{p} x_k \tag{2.27}$$

where $\frac{1}{p}$ is a small implementation term that scales up the retained activations. At evaluation time, the activations are passed as they are without any scaling, i.e., $\hat{x}_k = x_k$. The intuition behind Dropout is to force the network in learning more robust features that useful in conjunction with many different random subsets of units.

L2 regularization on the other hand, is another common form of regularization that complements Dropout. The idea behind it is to penalize the network for having large weights and it is implemented by adding the magnitude of all the weights to the network loss as in equation 2.28,

$$\mathcal{L}_{total}(\boldsymbol{W}) = \mathcal{L}(\boldsymbol{W}) + \frac{1}{2N} \lambda |\boldsymbol{W}|^2 \tag{2.28}$$

where N is the number of training examples and λ is the regularization constant. The intuitive interpretation of this regularization is to prevent the peaky weights and thus allow the network to use all of its inputs.

2.6.4 Batch Normalization

Batch normalization (BN) [78] is a recently developed technique that prevents the problems caused by poor weight initialization. It forces the activations throughout the network to take the form of a unit gaussian distribution and thus proposes a deterministic and normalized information flow. Considering a batch of data in the form of

$\mathcal{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with m instances, the normalization during training is done as in equation 2.29,

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}^{(i)},\ \boldsymbol{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2,\ \hat{\boldsymbol{x}}^{(i)} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \tag{2.29}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ would also take place in the backpropagation process. Although this normalization of activations allows efficient training, it is neither necessary nor desirable during evaluation time. Because of this, BN keeps a moving average of the activation means and variances during the training process as in equation 2.30,

$$\mathbb{E}^{moving}(\boldsymbol{x}) \leftarrow \mathbb{E}_{\mathcal{B}}(\boldsymbol{\mu}),\ \mathrm{Var}^{moving}(\boldsymbol{x}) \leftarrow \frac{m}{m-1}\mathbb{E}_{\mathcal{B}}(\boldsymbol{\sigma}^2) \tag{2.30}$$

where $\mathbb{E}_{\mathcal{B}}(\boldsymbol{\mu})$ is the expectation of $\boldsymbol{\mu}$ over multiple training batches $\mathcal{B}$ and $\frac{m}{m-1}\mathbb{E}_{\mathcal{B}}(\boldsymbol{\sigma}^2)$ is the expectation of the unbiased variance estimate over multiple training batches $\mathcal{B}$. These moving averages are then used for normalizing the activations during evaluation time as in equation 2.31.

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \mathbb{E}^{moving}(\boldsymbol{x})}{\sqrt{\mathrm{Var}^{moving}(\boldsymbol{x}) + \epsilon}} \tag{2.31}$$

Another way to view BN is as doing preprocessing at every layer of the network. It should be noted that BN also has a slight regularization effect as it adds noise to the activations of the layers. However, one should not rely too much on this regularization effect and should augment it with other possible regularization methods when possible.

Lastly, one should be very careful when using Dropout and BN together as a recent study of Li et al. [79] shows that they do not play well together. In their study, they found that the disharmony is caused by inconsistent behavior of the neural variance during the switch (from train to evaluation) of the network's mode. To overcome the limitations caused by the combination, they propose to either use Dropout after all the BN layers or to modify the formula of Dropout in order to make it more robust to variance.

2.7 Transfer Learning

Due to the lack of availability of large amounts of labeled data and to the huge amount of time it takes to train a convolutional neural network (CNN), in practice, it is common to use CNNs that are already trained on large datasets as ImageNet [80]. These pretrained networks are then used for performing classification/regression on different problems. In fact, it is also common to use pretrained CNNs across different tasks. This transfering of learned weights between tasks is referred to as *transfer learning* and it usually takes the following forms:

- **CNNs as frozen feature extractors:** In this form, the convolutional layers of a pretrained CNN are taken and they are used as fixed feature extractors, i.e., they are not updated during the training process. The extracted features are then used as inputs to the following untrained layers to perform the desired task.
- **Fine-tuning CNNs:** This form uses pretrained CNNs as initial points in the training process and trains whole network. In this form, it is also possible to only fine-tune the higher level convolutional layers as the lower ones are common feature (edge, corner) extractors.

Which one of these forms to use mostly depends on the size of the new dataset and the similarity of this dataset to the original one. If the new dataset is similar to the original one and its size is small, fine-tuning is not a good idea as it may easily cause overfitting. However, if the dataset size is large, fine-tuning would be a good idea as overfitting is less likely to occur.

CHAPTER 3

LITERATURE SURVEY

In this chapter, we will first start by presenting the traditional and deep learning based methods for the general problem of single-target visual object tracking and then move on to the studies that incorporate additional attention mechanisms for adaptive feature selection. Afterwards, we will go over the studies for the specific task of visual face tracking in video surveillance. Lastly, we will discuss the similarities and differences of our proposed face tracker with the presented studies.

3.1 Visual Object Tracking

Before going into the details of the related work, it is useful to start with a definition. Object tracking is the task of continuously predicting a target's location from the incoming and previous sensory data. In this section, rather than the general problem of object tracking itself, we will focus on the related studies in the more specific problem of single-target visual object tracking where the incoming data is video streams and there is a single target in each frame. We will review them under two different groups as in Figure 3.1. Multi-target tracking is left out as the task in this thesis is to track single targets.

3.1.1 Traditional Methods

In this subsection, we will focus on the relatively traditional trackers that were used before the deep learning era in computer vision. We will review them under four different groups where the first three make use fixed appearance models and the last

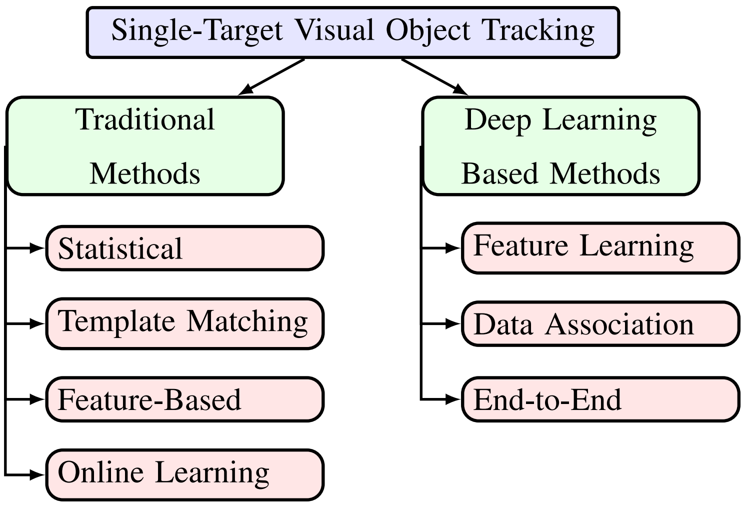


Figure 3.1: Methods for single-target visual object tracking.

one uses an adaptive one.

Statistical Methods: The Kalman filter [81] is probably the most well-known tracker; however, it works under the assumption that the probability density functions are Gaussian and the state transitions are linear. Although the Extended Kalman filter [82] enables it to work with non-linear state transitions, the Gaussian assumption still causes certain problems. To overcome these problems, particle filter approaches [83, 84] use Monte Carlo methods to allow working with non-Gaussian distributions.

Template Matching Methods: As the name suggests, template matching methods make use of the object's template for the tracking process. This is done by either using the correlation filters [85] or by searching for the maximum similarity [86, 87]. Additionally, in order to gain speed, the search is usually done in the neighborhood of the object's previous position.

Feature-Based Methods: As a classic feature tracker, the Kanade-Lucas-Tomasi (KLT) tracker [88] is one of the most commonly used tracker due to its efficiency and robustness to scale change. However, since it only uses the local information, it is prone to drifting errors.

Online Learning Methods: The previous three methods presented above, work with appearance models that are fixed. This causes certain issues on the performance since challenges as illumination, pose and scale changes and partial occlusions can change in the appearance of the target in a significant way. In contrast, online learning methods use adaptive appearance models (AAM) to keep updating their appearance models and thus perform better under these tough changes in tracking.

One of these methods with an AAM is the Incremental Visual Tracking (IVM) method proposed by Ross et al. [22]. In this study, eigenbasises are used in representing the targets and particle filters are employed for finding the best matching window. After a predefined number of windows, the eigenbasises are updated with the incremental principal component analysis method.

Another online learning method is the Tracking-Learning-Detection (TLD) method proposed by Kalal et al. [89]. TLD integrates the tracking and detection processes and thus can perform long term tracking. The estimation of the target locations are done by integrating the predictions of both the detector and tracker, and the detector is updated by the positive and negative samples around the target. If the tracker fails by some reason, it gets reinitialized by the detector.

Similar to the above online learning methods, Wang et al. [90] proposed the Discriminative Sparse Coding-Based Tracking (DSCT) method. DSCT uses sparse coding to to represent the targets. After the sparse codes of the positive and negative patches are obtained, they are passed through a linear classifier to identify the target in the future frames. To account for the appearance changes, it accumulates the most recent frames to build an AAM and uses it together with a static model to perform tracking.

Lastly, it should be noted that none of the object trackers presented in this section make use of learned rich hierarchical features that can serve well in representing and tracking objects. Because of this, they do not perform well in today's modern visual object tracking challenges [18–21] and thus they have fallen out of favor in the object tracking community.

3.1.2 Deep Learning Based Methods

This subsections focuses on the trackers that use deep learning methods that make use of the learned rich hierarchical features rather than the relatively traditional ones.

Although we review these studies under three different categories, drawing explicit lines between some of these approaches is very hard as they are tightly interconnected. Interested readers can refer to the review paper of Li et al. [91] for a more thorough review of the recent deep learning based trackers.

Before moving on to the review, it should be noted that although this subsection provides a review on the studies in deep learning based single-target object tracking, a general review of the studies in deep learning based object tracking including multi-target tracking methods and future target state prediction methods are out of the scope of this thesis. Interested readers can find more information on these areas in the recent survey paper of Krebs et al. [92].

Deep Learning for Feature Learning: The first deep learning based approaches treat deep neural networks as black-box feature extractors and aim to use the powerful features that are extracted from them. After the feature extraction process, they are used as input to subsequent traditional tracking methods for tasks as classification, association and filtering.

One of the first deep learning based approaches in visual object tracking is the Deep Learning Tracker (DLT) introduced by Wang and Yeung [93]. In this study, a Stacked Denoising Autoencoder (DAE) is trained in an offline manner using a large dataset to obtain generic feature representations. After the offline training phase, the encoder of the Stacked DAE is used as a pretrained feature extractor for a sigmoid classification layer in the online tracking phase. During tracking, in each frame, a set of particles are drawn from the estimated patch and their confidences are calculated using the network with a sigmoid output. If the sum of these confidences falls below a certain predefined threshold, the network is re-tuned.

Ma et al. [94] investigated a different approach by using both the higher and lower level features of a pretrained CNN called VGG-Net [5]. They argue that while the higher level features of the CNN encode semantic information, their spatial resolution is too coarse for precise target location inference. Because of this, they also use the feature maps from lower level layers as they contain more precise position information and perform the tracking using multiple layers. In their approach, the outputs of these layers are used as multi-channel features to learn an adaptive correlation filter per layer. After this learning process, the features maps of the VGG-Net are convolved with the learned correlation filters to obtain a response map and then by using this map the target location is inferred.

Similar to Ma et al., Wang et al. [95] also investigated the effect of using both the higher and lower level features and came up with the same findings. Further, they showed that only certain feature maps in a layer are relevant for tracking and introduced a feature map selection method that avoided the irrelevant features. The selected feature maps are then passed to two separate networks in the Fully Convolutional Network Based Tracker (FCNT) to infer the final target location.

Deep Learning for Data Association: One of the biggest difficulties faced in visual object tracking is the association of the target and its location in the current frame. Rather than using non-deep subsequent classification and association parts, some deep learning based approaches employ Siamese Networks [96] to learn a similarity measure and thus perform the task of data association.

One of the first trackers that used Siamese Networks for the task of data association is the Siamese INstance Search Tracker (SINT) introduced by Tao et al. [97]. In this study, the SINT is trained using the search and the query streams. The query stream gives a video frame with the exact position of the object to be tracked, whereas the search stream gives another frame with randomly sampled object locations. If these random object locations exceed a certain overlap threshold with the desired target location, they are considered to be positive training samples, otherwise they are used as negative samples. During tracking, the initial frame is constantly used for template extraction to measure the similarity of new candidate regions. After the similarity measure, the region with the highest similarity is considered to be the new target location.

Another data association approach, which won the VOT2017 real-time challenge [20], is the Siamese Fully-Convolutional (SiamFC) tracker introduced by Bertinetto et al. [15]. In this approach, a Siamese Network is used for learning a similarity map between an exemplar image and a candidate image. In order to cope with the possible different candidate image sizes, the Siamese Network is implemented as a fully-convolutional network. During tracking, a search image is passed to the network and the maximum score on the map is used in calculating the displacement of the target from frame to frame. Due to its real-time tracking ability (58-86 FPS), the SiamFC tracker has an advantage to be used for real world applications.

Also inspired by the Siamese CNNs, Li et al. [17] introduced the Siamese Region Proposal Network (SiamRPN) and won the VOT-RT2018 challenge [21]. In their approach, a Siamese Network is used for feature extraction and a region proposal network (RPN) is used in proposal generation. One of the branches in the RPN is responsible for foreground-background classification and the other one is used for proposal refinement. The whole system is trained in an end-to-end fashion. In the tracking phase, the tracking task is formulated as a one-shot detection task. Because of this, one of the most important properties of this SiamRPN tracker is that it can run at 160 FPS which makes it one of the fastest deep learning based trackers available today.

Deep Learning for End-to-End Tracking: The success of the end-to-end approaches in the area of computer vision is mostly due the automation of the whole pipeline. End-to-end approaches are also used in the visual object tracking community to automate the whole tracking pipeline, i.e., object representation - object extraction - location estimation. It should be noted that these approaches are different from the previous two in the sense that whole pipeline is jointly learned.

One of the first studies that used deep learning in an end-to-end fashion for the task of visual object tracking is the study of Gan et al. [98] that used of a recurrent neural network (RNN) for tracking. The RNN is trained in a fully offline manner with synthesized data that simulates moving objects. During training, the ground-truth location labels are also used along with the synthetic video frames. While tracking, the RNN outputs the bounding box annotation of the object in each frame.

Another end-to-end approach, which won the VOT2015 challenge [18] according to the expected average overlap score, is the Multi-Domain Network (MDNet) introduced by Nam and Han [13]. In this approach, the learning of domain-specific and domain-independent information is separated out. More specifically, a CNN with lots of domain specific binary classification branches is first pretrained offline by using a large set of videos to obtain a generic target representation in the shared network, which accounts to domain-independent information. When tracking in a new sequence, a new binary classification branch is added to the network and it is updated in an online manner. The tracking is performed by evaluating candidate windows that are randomly sampled around the previous target's state. One major disadvantage of this online tracker however, is that its speed (1 FPS) is too low for real-time applications.

A rather different approach which uses two pretrained CNNs (AlexNet [99]) and a regression network was investigated by Held et al. [14] under the name Generic Object Tracking Using Regression Networks (GOTURN). The GOTURN tracker is also trained in a completely offline manner using both real and synthetic data. In the training phase, two consecutive video frames are fed to the two pretrained CNNs, and the concatenated outputs of these CNNs are used as input to a fully-connected regression network for regressing the current frame's bounding box. After the network is trained, during tracking, the GOTURN tracker outputs the current frame's target location and uses it to obtain the search region for the subsequent frame. A really important property of this tracker is its ability to track objects up to 100 FPS, making it one the fastest deep learning based tracker for single-target tracking present today.

Inspired from the real-time regression-based object detection approach of You Look Only Once (YOLO) [100], Ning et al. [101] introduced the Recurrent YOLO (ROLO). In this approach, the high-level features of a pretrained CNN and bounding box regressions of the YOLO object detector are fed as input to an RNN to model the spatio-temporal information. The RNN is trained in an unsupervised manner to predict the next bounding box annotation. The authors show that the spatio-temporal modeling ability of ROLO enables it to tackle major occlusions and severe motion blurs during the harsh tracking process.

3.2 Attention Mechanisms in Visual Object Tracking

In visual object tracking, different features may have different effects in tracking different objects. Using all of the feature is neither efficient nor effective. Because of this, several adaptive feature selection methods have been developed in the object tracking literature. In this section we will briefly go over them.

Two of these adaptive methods are the Structuralist Cognitive Model for Tracking (SCT) by Choi et al. [102] and the Attentional Correlation Filter Network (ACFN) by Choi et al. [103]. In these approaches, an attention network is used for selecting the best subset of correlation filters for the object to be tracked.

Rather different approaches which incorporate attention mechanisms in recurrent neural networks (RNN) are the Recurrent Attentive Tracking Model (RATM) by Kahou et al. [104] and the Hierarchical Attentive Recurrent Tracker (HART) by Kosiorek et al. [105]. Both of these approaches use attention mechanisms to separate the where and what processing pathways to suppress the negative effect of the extracted irrelevant features. The attentions used in these approaches are soft attention mechanisms and they are learned by using gradient-based methods.

Recently, Hu et al. [106] demonstrated the importance of channel-wise attention mechanisms for the task of image recognition. Inspired by this, He et al. [107] combines this channel-wise attention mechanism with the SiamFC tracker [15] and proposes the Semantic Appearance Siamese network (SA-Siam) for real-time object tracking. In SA-Siam, the channel attention module plays an important role by giving higher weights to the channels that have an importance in tracking specific objects. Another tracker that also incorporates channel attention and other kinds of attentions is the Residual Attention Siamese Network (RASNet) proposed by Wang et al. [108]. In this approach, several attention mechanisms as general attention, residual attention and channel attention are used for adapting the offline trained model, without updating the model in an online manner.

3.3 Visual Face Tracking in Video Surveillance

Visual face tracking is a special case of visual object tracking where the target is a face of an individual or a group of individuals. In this section, we will focus on the studies that perform visual face tracking under surveillance conditions where there are lots of variations in pose, scale, expression and illumination and occlusions in cluttered scenes.

Face tracking in the context of video surveillance is usually done by using the relatively traditional methods presented in Section 3.1.1. For instance, Dewan et al. [1] compares three different generic adaptive appearance modeling trackers – Incremental Visual Tracking (IVT) [22], Tracking-Learning-Detection (TLD) [89] and Discriminative Sparse Coding-Based Tracking (DSCT) [90] – with surveillance applications in mind and shows that IVT outperforms the other two in terms of tracking accuracy and computation time. They further argue that the low discrimination power of the TLD face descriptor and computational complexity of the DSCT are the main limitations of these two methods. However, it should be noted that although the IVT tracker performs the best, together with the other trackers, it does not satisfy the minimum real-time tracking requirement of 25 FPS.

Another traditional video surveillance study is the face tracking framework proposed Lan et al. [2]. In this framework, an already published face detector [109] and a histogram-assisted Kanade-Lucas-Tomasi [88] (HAKLT) tracker is integrated together to perform face tracking. The HAKLT tracker makes use of a color histogram to get rid of the drifting problems of the naive KLT tracker and it enables speeds that are above 100 FPS.

Although these traditional trackers can perform up to certain degrees, they do not make use of the learned rich hierarchical features present in deep learning based trackers and thus they are prone to failure under the harsh conditions present in video surveillance. The increasing usage of deep learning based trackers in the Visual Object Tracking (VOT) challenges [18–21] is already an indicator of their significant potential in object tracking tasks. It should be noted that the harsh conditions of video surveillance are also present in these challenges.

A thorough search of the literature had yielded no published study on using deep learning based methods for the task of visual face tracking under surveillance conditions. However, there are a few studies [110–112] that incorporate deep learning methods for regular (not under surveillance conditions) face tracking. In the rest of this section, we will briefly go over them to present how deep learning is utilized in face tracking.

One of these few deep learning based face trackers was introduced by Ren et al. [110].

In this study, a lightweight and coarse-to-fine convolutional neural network (CNN), which is inspired by the MTCNN face detector of Zhang et al. [113], is used for detecting faces in each video frame. Additionally, a Kalman filter [81] tracking approach is employed for cases where the faces get largely deflected or severely occluded. This tracker operates above 25 FPS; however, its speed is dependent on the resolution of the input video.

Another face tracker that uses deep learning is the Deep Manifold Embedding Active Shape Model (DME-ASM) proposed by Choi and Kim [111]. In this tracker, a CNN is used for categorizing the corresponding pose range for the input face. This tracker is specifically build for solving the diverse head pose problem in face tracking and it can operate in real-time.

A different approach which uses deep face tracking as a part of automatic labeling of faces for films and TV material is the study by Parkhi et al. [112]. In this study, a tracking-by-detection approach is utilized. Faces are first detected using a local version of the cascaded Deformable Part Model (DPM) [114] and then a KLT [88] tracker is used in grouping these detections through consecutive frames. However, rather than the tracking task itself, a CNN is only used for the task of face track classification.

Lastly, it should be noted that none of the deep face trackers presented above use deep learning to specifically perform tracking, i.e., they only use it for assisting the relatively traditional tracking methods. Thus, they do not use deep learning in an end-to-end manner, i.e., they do not solely take the frames as input and output the bounding box annotations.

3.4 The Place of Our Tracker

In this thesis, we choose to take the real-time GOTURN tracker [14] as our starting point due to its high performance and simple end-to-end form, i.e., it just takes the consecutive two frames as raw input and outputs the bounding box annotation for the query frame. Thus our tracker falls under the category of end-to-end deep learning based object trackers presented in Section 3.1.2. Another very important property of the GOTURN tracker is its ability to run at speeds above 100 FPS (165 FPS when run on more advanced hardware) which is due to its fully offline training procedure.

After choosing the GOTURN tracker as our starting point we extend it using several useful extensions. One of the extensions is the usage of both the lower and higher level features in the tracking process as in the FCNT tracker [95]. However, differently, we make use of these features in an end-to-end manner. Another extension is the usage of a fusion network in the regression network which is adapted from the studies of Akkaya and Halıcı [115, 116]. We also use a channel-wise attention mechanism as in the case of the SA-Siam tracker [107]. However, rather than using it for just the static channels of the last two layers, we use it to adaptively weight the dynamic channels from all of the layers. We also learn the attention weights in an end-to-end manner rather than using a multi-step training procedure. The details of these extensions will be presented in Section 4.1.

In contrast to the relatively traditional video surveillance trackers [1, 2] presented in Section 3.3, our tracker makes use of the learned rich hierarchical features of deep neural networks and runs at speeds very far beyond 25 FPS for real-time tracking. It should be noted that there is no hand-engineering involved at any stage, i.e., we just feed the frames as input to the network and get the bounding box annotations as the output. Also different from the deep learning based face trackers that are not used for tracking under surveillance conditions [110–112], presented again in Section 3.3, our face tracker uses deep learning for the tracking task itself. This is opposed to using it in the other modules and employing relatively traditional trackers in the tracking module.

CHAPTER 4

PROPOSED METHOD

This chapter starts by describing the details of the proposed end-to-end attentive deep network architecture and then moves on to the details of the employed soft channel attention mechanism. Afterwards, we present the details of the offline training and online tracking procedures.

4.1 Network Architecture

Inspired from the single-target Generic Object Tracking Using Regression Networks (GOTURN) tracker of Held et al. [14] and the studies described in Section 3.4, we propose a face tracking network named Attentive Face Tracking Network (AFTN) whose architecture is as in Figure 4.1. The input to the network is a pair of 224$\times$224 RGB images that are cropped from the previous and current frames that act as the target object and the search region respectively. The output is a tuple with three elements that describe the location and size of the target's bounding box within the 224$\times$224 search region. Importantly, the training of AFTN is done in a fully end-to-end manner.

In more detail, the proposed AFTN is composed of the following four subnetworks:

- **Feature Extraction Network (FEN):** The FENs are composed of two pre-trained VGG-Face [117] networks (FEN-p and FEN-c) that act as frozen feature extractors for the previous and current input frames. It should be noted that only the convolutional blocks of the VGG-Face network are used and they are not trained any further to prevent the possible overfiting cases due to the

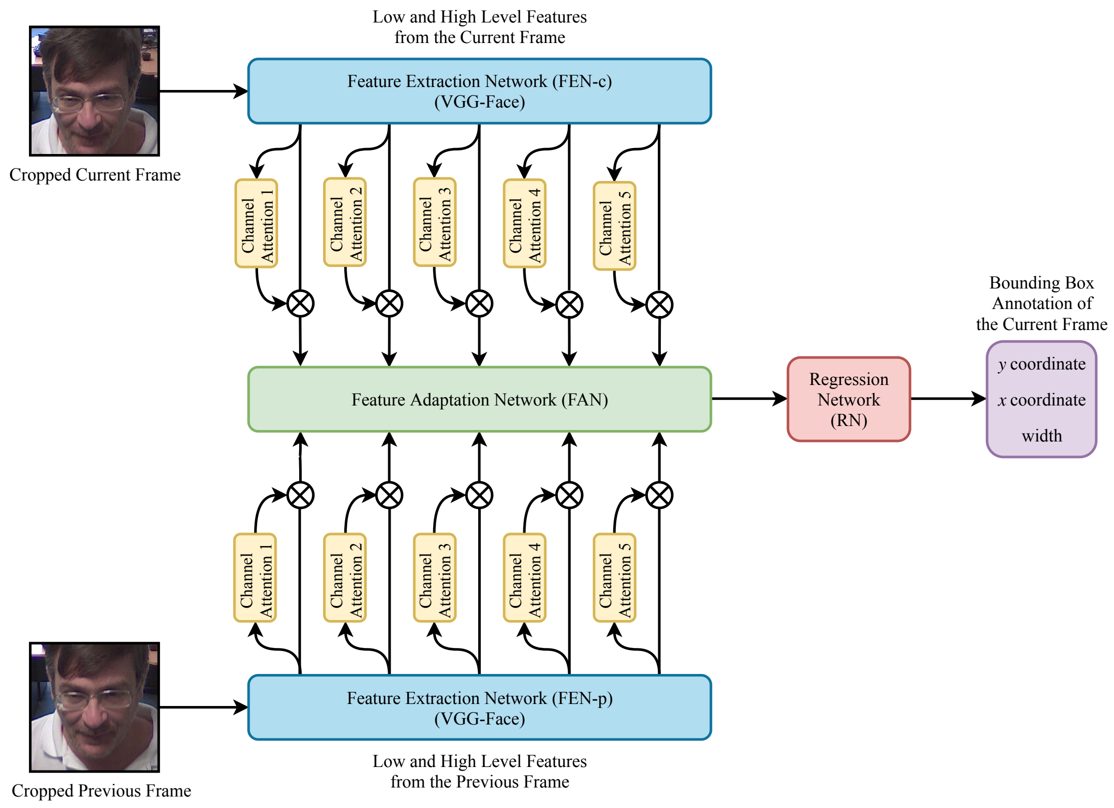


Figure 4.1: The network architecture of the proposed AFTN. After the features are extracted by the FENs, they are passed as input to the CANs to get weighted. After the explicit weighting, the weighted features are concatenated in the channel dimension via the FAN and then they are passed to the RN for regressing the bounding box annotations.

relatively small size of the dataset. While the shallow layers of the FENs extract simple low-level features as edges and corners, the deeper layers extract more complex high-level features as the semantics of the target. Although these high-level features are very useful in tasks that require semantic information, their receptive fields in the input image are very large making them less precise in localizing the targets in object tracking. Therefore, both the high-level and low-level features of the FENs are used together. Since it is not immediately clear which level features will serve well in the tracking process, we basically use all the extracted features from all the layers and provide them as inputs to the channel attention networks to explicitly weight their channels.

- **Channel Attention Network (CAN):** The CANs are simple two layer multilayer perceptrons (MLP) with single sigmoid outputs corresponding to the weight coefficients of the input channels. These coefficients are then used for weighting the channels according to their importance in the tracking process. After the weighting process, the weighted channels are passed to the feature adaptation network for concatenation. Since we use the 5 different layers from the FENs, there are also 5 different CANs. However, opposed to the FENs, these CANs are trained during the offline training process.

- **Feature Adaptation Network (FAN):** The FAN is used for concatenating all the weighted features coming from the CANs in the channel dimension. It has no learnable parameters and thus just serves as a basic feature concatenator. The concatenated features are then passed to the regression network to perform the final regression step.

- **Regression Network (RN):** The RN is composed of a fusion network (convolutional layer with 1×1 kernels) followed by a three layer MLP and it is used for regressing the target's bounding box annotation from the concatenated input features. The fusion network is used for fusing the concatenated features coming from the FAN. As in the case of CANs, the RN is also trained during the offline training process.

It should be noted that although this section provided a higher level view of the proposed network architecture, it does not contain any information regarding the imple-

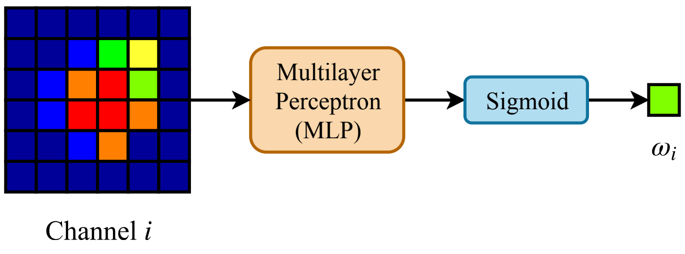


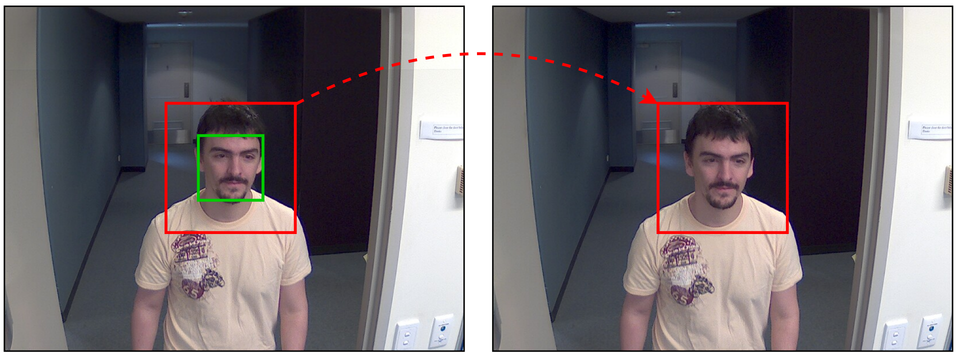
Figure 4.2: The soft channel attention mechanism that generates the weighting coefficient ω_i for the i-th channel. After Channel i is passed through an MLP with a sigmoid output, it gets weighted by its own weight coefficient ω_i and then it is passed as input to the next module in the tracker.

mentation details as the number of layers, the number of neurons in the layers, the employed regularization layers etc. These implementation details will be given in Section 5.3 while describing the test network architectures.

4.2 Channel Attention Mechanism

Different channels play different roles in tracking different targets. While some channels may be very important in tracking certain targets, they might not effect the performance at all in tracking other ones. If we could adapt the channel importance to the target, we would achieve great results in tracking them. However, it is not immediately clear which of the channels will serve well in tracking certain targets. In order to automate this channel selection process, we propose to use a soft channel attention mechanism whose details is as in Figure 4.2.

More specifically, we first flatten the 6×6 channels and then pass them through multilayer perceptrons (MLP) with sigmoid outputs to obtain their weight coefficients. The sigmoid functions have a bias of 0.5 to ensure that no channel will be suppressed down to zero and each channel i has a weight coefficient ω_i associated with itself. Since the lower layers in the feature extraction network (FEN) have channels with greater sizes as 54×54 and 13×13, we max-pool them to match the 6×6 size of the last layer and then pass them through the attention mechanism. After the weight coefficients are obtained, the channels are multiplied with their corresponding weights



Previous Frame ($t-1$) Current Frame (t)

Figure 4.3: Two random consecutive frames from the ChokePoint dataset. The green box is the ground-truth bounding box of the previous frame and the red boxes, twice the size of the green one, are the regions that are cropped, resized and then fed into the network.

and then they are passed as input to the next module for concatenation. It should be noted that the MLPs share weights across the channels that are extracted from the same convolutional layer.

4.3 Offline Training

In the offline training phase, we randomly choose a pairs of successive frames from our training dataset with a batch size of 50. We then crop these pair of frames using the previous frame's bounding box annotation and resize them to 224×224 to match the input size of the VGG-Face [117] network. More specifically, the bounding box of the previous frame is expanded to twice its size and this region is cropped from both of the frames as in Figure 4.3. We also subtract the mean of the dataset that was used in training VGG-Face. Since we feed the cropped images to the network, we also transform the bounding box annotations of the 800×600 images to the 224×224 ones by simple linear transformations and we use these transformed annotations for the training process. In more detail, the bounding box annotations are first computed with respect to the cropped regions and then they are scaled by $\frac{224}{\text{width of cropped region}}$ to match the 224×224 input images.

After the preprocessings, we feed the data batch to the network that is initialized using the He Initializer [76] and compute the L1 loss between the ground-truth annotations and the predicted ones multiplied by 10. We use this scaling by 10 to regress lower numbers which is better for network training given that the biases are initialized at zero. We then backpropagate from this loss and use the Adam optimizer with a learning rate of $1e-5$ to optimize the parameters of the network. The training is done for 10 epochs and the PyTorch automatic differentiation framework [46] is used in the implementation process.

Lastly, it should be noted that only the channel attention networks (CAN) and regression network (RN) are trained in the offline training phase as the feature extraction network is already pretrained and the feature adaptation network has no learnable parameters. Another important thing to note is that data augmentation methods are not used as they had no effect in the tracking performance.

4.4 Online Tracking

In the online tracking phase, first, the initial two frames are read from the video sequence and the regions that contain a face are cropped from both of them using the same procedure described in the previous section. After the cropping, both of the crops are resized to 224×224 to be compatible with the VGG-Face [117] network. The mean of the dataset that the VGG-Face network was trained with is again subtracted. Then, after all these preprocessings, both of the frames are fed into the network to obtain the bounding box annotation of the current frame. Specifically, the (x, y) coordinates of the top-left corner together with the width (height) of the current frame's bounding box in the 224×224 search image is obtained. This bounding box is then transformed back to its corresponding location before the resizing in order to find the face's actual location in the 800×600 frame. In more detail, this transformation is done by first multiplying the network's prediction by 10, then scaling this prediction by $\frac{\text{width of cropped region}}{224}$ and finally adding the (x, y) coordinates of the top-left corner of the cropped region to the first two elements of the network's prediction.

After the actual target location of the current frame is obtained, the current frame is used as the previous frame and a new frame is read to be the current frame. Using the predicted bounding box annotation from the previous step, the frames are cropped-resized again and they are again fed to the network to regress the target bounding box in the newly read frame. This crop-resize-feed-read prodecure continues for the rest of the video frames and by this way, the tracker tracks the faces until the end of the sequence.

CHAPTER 5

EXPERIMENTAL RESULTS AND DISCUSSION

This chapter starts by describing the face dataset that is used in this study. We then move on to details of the evaluation metrics that are used in comparing the trackers. After the description of the evaluation metrics, we continue by presenting the details of the test network architectures and their testing procedures. We then present the quantitative accuracy, robustness and speed results using several informative plots and tables. Lastly, in order to visualize the effect of the channel-wise attention mechanism, we provide a qualitative analysis.

5.1 Dataset

In order to test our proposed face tracker, we use the publicly available ChokePoint dataset [23] that was sponsored by National ICT Australia (NICTA) Limited. This dataset was originally designed for experiments in person identification/verification under real-world surveillance conditions. However, it can also be used for different tasks as face-quality measurement, pedestrian/face tracking and person re-identification.

The ChokePoint dataset was recorded using 3 different cameras (C1, C2, C3) placed above several portals, which are natural choke points in pedestrian traffic, in order to capture subjects walking through the portals in a natural way. (see Figure 5.1) The faces captured by these three cameras vary in illumination conditions, pose, sharpness, as well as misalignment. Due to this 3 camera configuration, in each sequence, only one of the cameras is able to capture the faces near-frontal through a certain period of time.

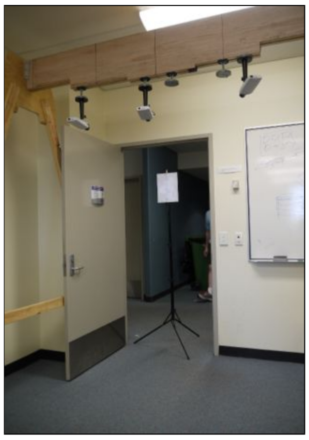


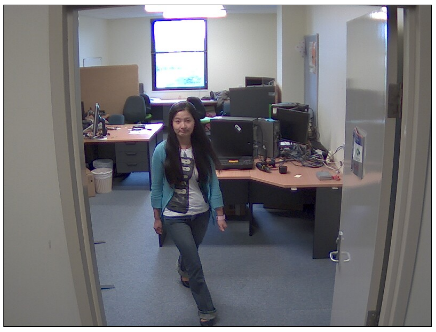
Figure 5.1: The original recording setup used for the ChokePoint dataset in the portal 1 entering (P1E) scenario. 3 different cameras are used for recording the entry of a subject from 3 different viewpoints. This 3 camera setting allows near-frontal face capture by at least one of the cameras. Source: [118]

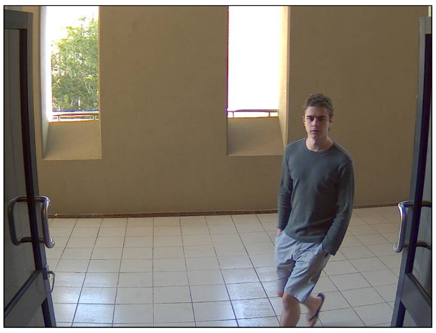
The dataset consists of 25 subjects (19 male and 6 female) in portal 1 (P1) and 29 subjects (23 male and 6 female) in portal 2 (P2). The frame rate is 30 FPS and the resolution of each frame is 800×600. In total, the dataset consists of 48 different video sequences and 64,204 frames with a face. Importantly, at a given time, only one of the subjects is present in a frame.

Each sequence was named after its recording conditions where P, S, C stand for portal, sequence and camera respectively. E and L indicate whether if subjects are entering or leaving the portal. The numbers indicate the portal number, sequence number and camera number respectively. For instance, P1E_S2_C3 indicates that the video sequence was recorded using camera 3, at portal 1, when subjects were entering in sequence 2. Example shots from the dataset with various backgrounds can be seen in Figure 5.2.

For the evaluation, a baseline verification protocol [23] is designed for the ChokePoint dataset. In this protocol, video sequences with near-frontal face views are divided into two distinct groups, namely G1 and G2 (see Table 5.1), where each group plays the role of development and evaluation set in turn. The parameters are first learned on the development set and then they are applied to the evaluation set. The average performance measures on the evaluation sets are then used for reporting the final

(a) P1E_S1_C1


(b) P1L_S1_C1


(c) P2E_S2_C2

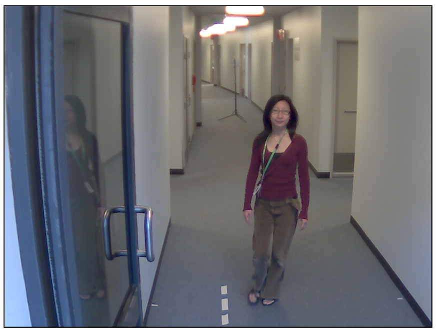

(d) P2L_S3_C3

Figure 5.2: Example shots from the ChokePoint dataset that show entering and leaving scenarios from the two portals with various backgrounds. While the recording environments of P1E, P1L and P2L are indoor, P2E is recorded outdoor. Source: [118]

Table 5.1: The two distinct groups G1 and G2 in the baseline verification protocol for the ChokePoint dataset.

G1	P1E_S1_C1	P1E_S2_C2	P2E_S2_C2	P2E_S1_C3
	P1L_S1_C1	P1L_S2_C2	P2L_S2_C2	P2L_S1_C1
G2	P1E_S3_C3	P1E_S4_C1	P2E_S4_C2	P2E_S3_C1
	P1L_S3_C3	P1L_S4_C1	P2L_S4_C2	P2L_S3_C3

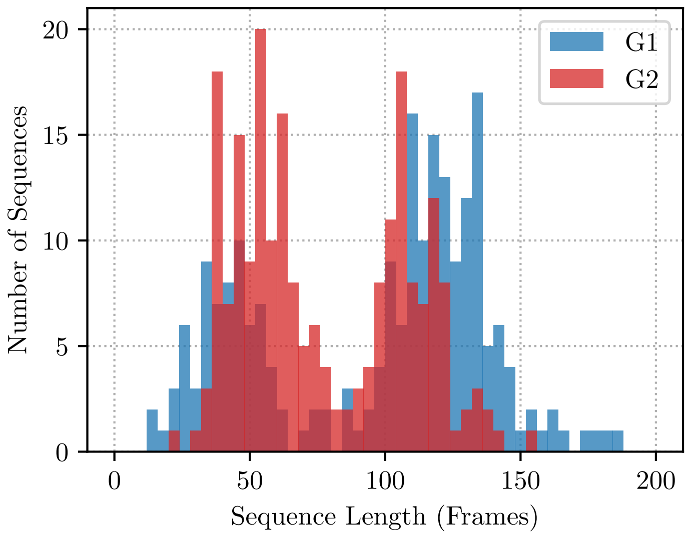


Figure 5.3: Histogram for the video sequence lengths (in frames) in G1 and G2. The number of bins is 50. G1 has an average of 95.6 and G2 has an average of 77.1.

results.

Lastly, the annotations provided with the ChokePoint dataset are the eye locations and the person IDs in each frame. However, for face tracking, or any other kind of visual object tracking, the bounding box annotations are required.

In this thesis, we annotate[1] the frames in the G1 and G2 sets with bounding boxes and thus make the ChokePoint dataset available for visual face tracking. We discard the frames where there is no face of a person. This newly formed dataset again consists of 25 subjects (19 male and 6 female) in P1 and 29 subjects (23 male and 6 female) in P2. In total, the dataset consists of 432 different video sequences (216 in G1 and the other 216 in G2) each having a single person present at a given time and 37,307 frames (16,665 in G1 and 20,652 in G2) with a face. The average length of a video sequence is 95.6 frames for G1 and 77.1 frames for G2. (see Figure 5.3)

The bounding box annotations are in the form of a tuple with three elements: the y coordinate of the top-left corner, the x coordinate of the top-left corner and the width

[1] We perform the annotation by first using a face detector, which is built using the dlib toolkit [119], to detect the faces and then we manually go over the bounding boxes to correct for the mistakes that the detector makes. It should be noted that the annotations for the faces could have also been extracted from the provided eye location labels by using some face ratio heuristics; however, considering the variances in human face ratios and the change of the person's angle to the camera, this can easily lead to incorrect annotations.

(or height) of the bounding box. It should be noted that all these annotations are with respect to the 800×600 frames.

Finally, for the evaluation, we use the baseline verification protocol for the ChokePoint dataset, i.e., we first use G1 to train our network and use G2 to test it and then do the reverse. At the end, we report the average performances of our trackers on the evaluation sets.

5.2 Evaluation Metrics

In order to evaluate the performance of a single-target object tracker, several evaluation metrics are used in the literature. However, there is no single agreed metric. Čehovin et al. [120] attempts to bring a consensus to this absence of homogenuity by showing that lots of the seemingly different metrics are actually highly correlated. In their study, they use 25 widely used video sequences and 13 different trackers, and show that a single-target tracker's performance can be boiled down to two complementary measures, namely accuracy and robustness.

This section will continue by summarizing the commonly used evaluation metrics in the literature and then it will explain the two complementary metrics proposed by Čehovin et al. that are also used in this thesis.

Following the study of Čehovin et al., we first start by defining an object state description Λ in a sequence with length N as in equation 5.1,

$$\Lambda = \{(A_t, \boldsymbol{x}_t)\}_{t=1}^{N} \tag{5.1}$$

where A_t is the region of the object and $\boldsymbol{x}_t \in \mathbb{R}^2$ is the center of the object at time t. In practice, A_t is usually a bounding box that is either a square or a rectangle. An ideal evaluation metric should be able to summarize how the predicted object state Λ_T matches with the ground-truth object state Λ_G.

5.2.1 Center Error

Center error δ_t is one of the oldest evaluation metrics in the literature. It measures the absolute distance between the predicted target's center $\boldsymbol{x}_t^T$ and the ground-truth center $\boldsymbol{x}_t^G$ as in equation 5.2.

$$\delta_t = ||\boldsymbol{x}_t^G - \boldsymbol{x}_t^T|| \tag{5.2}$$

The overall error is either shown as a center error vs. frame plot or summarized as an average error (see equation 5.3) or as a root-mean-squared error (see equation 5.4).

$$\Delta_\mu(\Lambda^G, \Lambda^T) = \frac{1}{N}\sum_{t=1}^{N} \delta_t \tag{5.3}$$

$$\text{RMSE}(\Lambda^G, \Lambda^T) = \sqrt{\frac{1}{N}\sum_{t=1}^{N} ||\boldsymbol{x}_t^G - \boldsymbol{x}_t^T||^2} \tag{5.4}$$

Although it requires a minimal annotation effort, which is just a point, the drawback of this metric is that it is not immediately clear where the center of the object is. Additionally, it ignores the target's size which in turn makes it misleading. For instance, large targets may have large center errors compared to the small ones; however, this error may be very small compared to their sizes. To remedy this, the normalized center error $\hat{\delta}_t$ is used instead. In this error, the center error is normalized by using the ground-truth target size as in equation 5.5.

$$\hat{\delta}_t = \left|\left|\frac{\boldsymbol{x}_t^G - \boldsymbol{x}_t^T}{\text{size}(A_t^G)}\right|\right| \tag{5.5}$$

Despite the normalization, the center error metric can still be misleading as a large target size may hide a large center error when the predicted target size and ground-truth are not close. Thus center error metrics are not considered to be good evaluation metrics as they may be very misleading.

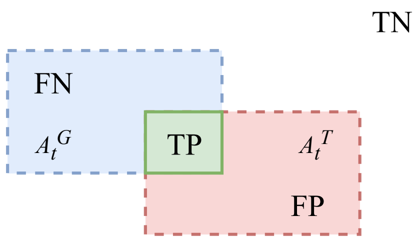


Figure 5.4: An illustration of the tracker's predicted region overlapping with the ground-truth region.

5.2.2 Region Overlap

Region overlap ϕ_t is the ratio of the intersection between the predicted target region A_t^T and the ground-truth region A_t^G to their union. It is illustrated in Figure 5.4 and calculated as in equation 5.6.

$$\phi_t = \frac{A_t^G \cap A_t^T}{A_t^G \cup A_t^T} = \frac{TP}{TP + FN + FP} \tag{5.6}$$

The overall region overlap over the whole sequence is summarized by either an average overlap or a ratio of correctly tracked frames to all of them given a certain threshold. The latter approach is adapted from the object detection community and it is called the true positive score P_τ (see equation 5.7). This score has become very popular in tracker evaluations with the tracking-by-detection concept.

$$P_\tau(\Lambda^G, \Lambda^T) = \frac{||\{t|\phi_t > \tau\}_{t=1}^N||}{N} \tag{5.7}$$

A nice property of the region overlap metric is that it simultaneously takes into account both the size and position of the predicted target region. Furthermore, it ranges between 0 and 1 as opposed to the center error which can become arbitrarily large in some situations.

5.2.3 Tracking Length

Tracking length is the number of successfully tracked frames from a trackers initialization to its first failure. A failure can be determined by using an overlap-based failure criterion, i.e., if the region overlap falls below a certain threshold τ, the tracker can be considered to be failed. In the following sections, the tracking length measure will be denoted as L_τ.

Although the tracking length explicitly addresses the tracker's failure, it has a significant drawback of being dependent on the initial conditions of the video sequence. If the initial frames of the sequence contains difficult frames to track, the tracker will easily fail and the rest of the video will be discarded. Due to this, the tracking length on its own is not considered to be a good evaluation metric.

5.2.4 Failure Rate

Failure rate is the ratio of the number of reinitializations, upon failure, to the number of frames in a video sequence. A failure can again be determined by using an overlap-based failure criterion as in the tracking length measure case. In the following sections, the failure rate measure with threshold τ will be denoted as F_τ.

Compared to the tracking length metric, the failure rate has the advantage of evaluating the tracker on the whole video sequence. This decreases the importance of the hardness of the initial frames. Thus, it serves as a better evaluation metric than the tracking length.

5.2.5 The Two Complementary Measures: Accuracy and Robustness

After comparing ten different evaluation metrics for single-target trackers, Čehovin et al. [120] show that some of the seemingly different metrics are actually highly correlated. The correlation matrix they obtained with this study is as in Figure 5.5. This correlation matrix clearly shows that metrics from 1 to 3 and from 4 to 7 are highly correlated. The first cluster (1-3) consists of center error based metrics and the second one (4-7) consists of overlap based ones. It can also be observed that there is a

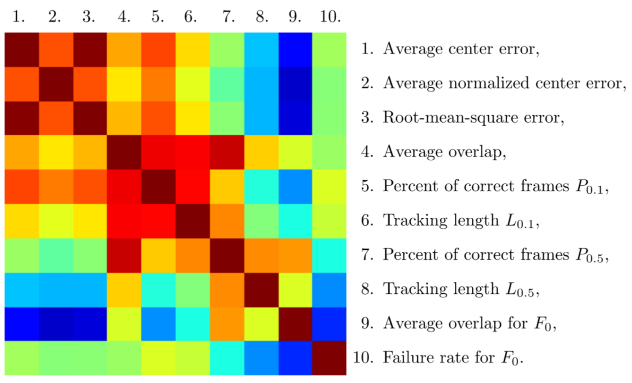


Figure 5.5: The correlation matrix for all of the evaluation metrics in the study of Čehovin et al. Red regions are higher in value.

slight overlap between the two clusters which implies similarity in their information content. Based on their analysis, Čehovin et al. conclude that average overlap is one of the most appropriate metrics to be used in tracker comparison as it is scale and threshold invariant and exploits the entire sequence. They propose to use the average overlap measure to account for the *accuracy* (A) of a tracker. They also take the failure rate metric itself as a measure of robustness and propose to use it to account for the *robustness* (R) of a tracker. By this way, they define a new A-R evaluation metric as a pair of complementary scores.

To better understand how these accuracy and robustness metrics complement each other, consider the two extreme theoretical trackers: T1 and T2, visualized in an A vs. R plot as in Figure 5.6. T1 is a tracker which always reports its initial condition for the whole sequence. Because of this, T1 will fail a lot resulting in a very low R value. However, because of the frequent reinitializations, it will be displayed near the top-left of the plot. T2 on the other hand, is another tracker which always reports the whole image to be the region of the object. Because of this, T2 will provide loose regions, but it will not fail. Therefore it will be displayed near the bottom-right corner. An ideal tracker on the other hand, would have to reach the top-right corner.

In this thesis, we will follow Čehovin et al. and compare different trackers using the complementary A-R evaluation metric. It should be noted that this A-R metric is also used in visual object tracking challenges [18–21]. However, we will use slightly different metrics to account for the accuracy and robustness. More specifically, rather

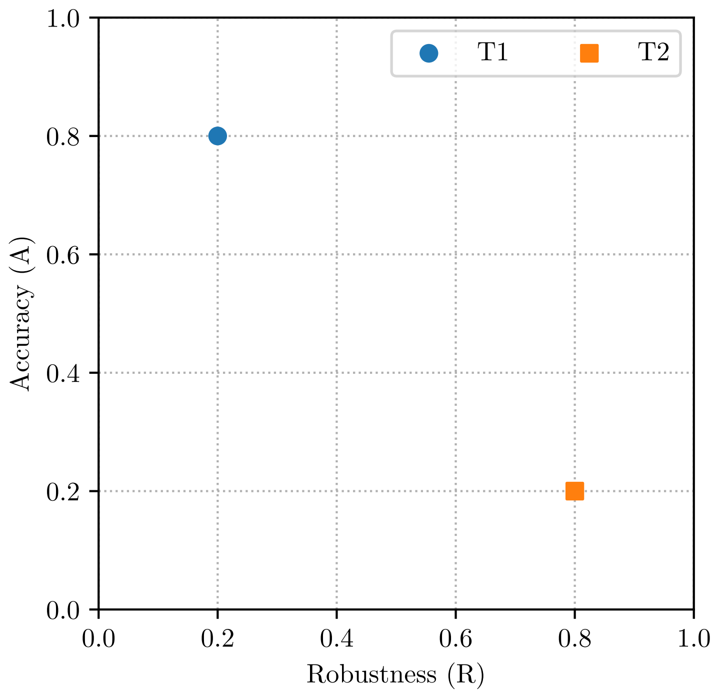


Figure 5.6: Two theoretical trackers visualized on an Accuracy vs. Robustness plot. The closer the tracker is to the top-right corner, the better it is.

than the average overlap, we will use the area under the curve (AUC) of the True Positive vs. Region Overlap Threshold (TP vs. ROT) plot (see Figure 5.7) to account for accuracy. And rather than the failure rate with $\tau = 0$, we will use the area above the curve (AAC) of the Failure Rate vs. Reinitialization Threshold (FR vs. RT) plot (see Figure 5.8) to account for robustness. It should be noted that the AUC of the TP vs. ROT plot is actually equivalent to the average overlap metric[2], and the AAC of the FR vs. RT plot is an average of the failure rates for different RTs rather than a failure rate metric with $\tau = 0$. It should be noted that, these two plots have the additional advantage of displaying the tracker's performances for not only one, but all of the thresholds values.

The TP vs. ROT plot for a single video sequence is obtained by first running a tracker over the whole sequence and calculating the ratio of frames where the region overlap is greater than a certain overlap threshold, and then by doing this for all thresholds ranging from 0 to 1. In this plot, the RT is 0, i.e., the tracker gets reinitialized only if it

[2] See the supplementary material of Čehovin et al. [120] for the proof.

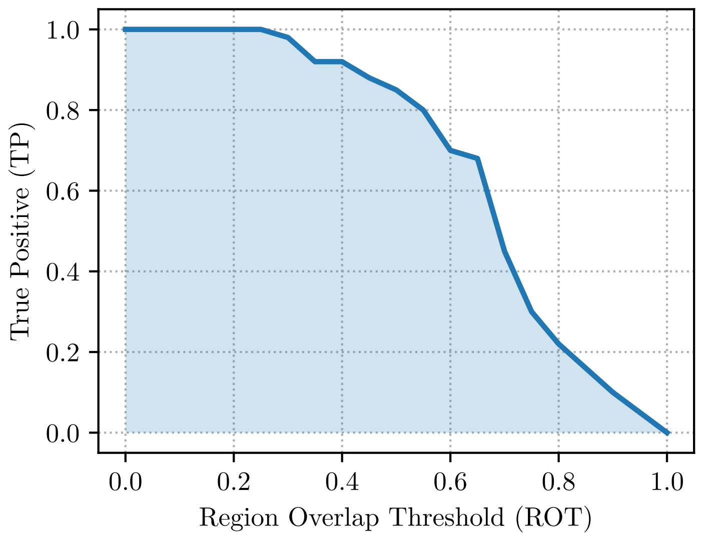


Figure 5.7: True Positive vs. Region Overlap Threshold plot of an arbitrary tracker. The filled blue area accounts for its accuracy.

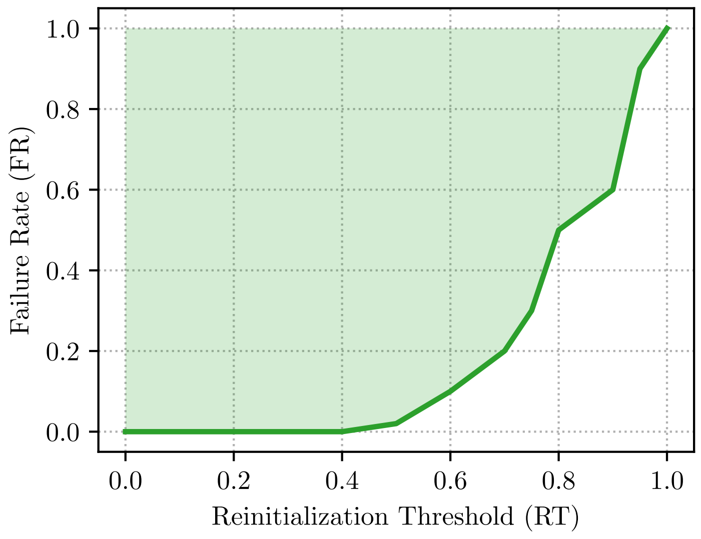


Figure 5.8: Failure Rate vs. Reinitialization Threshold plot of an arbitrary tracker. The filled green area accounts for its robustness.

completely looses the target. Similarly, the FR vs. RT plot for a single video sequence is obtained by first running the tracker over the whole sequence and calculating the ratio of frames where the region overlap failed below the RT, and then again by doing this for all thresholds ranging from 0 to 1. However, unlike the TR vs. ROT plot case, in the FR vs. RT plot, the tracker gets reinitialized immediately after the region overlap falls below the RT.

5.3 Test Network Architectures

In this section, we describe the details of the network architectures to be tested. In order to compare our proposed tracker with the other ones, we start by using the GOTURN tracker [14] as our baseline and then add/remove the necessary pieces one by one. In total, we test 10 different network architectures whose details are summarized in Table 5.2.[3] We use $C_A^X/cp - C^Y - F^Z$ as our tracker naming convention where A denotes whether if attention is used or not, X denotes the layers that are used in the feature extractor network (FEN) and Y and Z denotes number of convolutional and fully connected layers used in the regression network (RN). cp is used for the cases where both the current and previous frames are used as input to the tracker. If only the current frame is used, then only c is used.

Baseline Network: We first start by describing the architectural details of the GOTURN tracker that is used as our baseline. The GOTURN tracker takes the current and previous frames as input and passes them through the convolutional layers of a pretrained AlexNet [99]. After the feature extraction process, the output of the last convolutional layer is first flattened and then passed through a 4 layer multilayer perceptron (MLP) for regressing the bounding box annotation of the current frame. The first 3 layers of the MLP has 4096 neurons with ReLU activations and 0.5 dropout, and the last layer has 4 neurons that correspond to the (x, y) coordinate of the top-left corner and to the height and width of the bounding box.

In this study, rather than using a pretrained 5 layer AlexNet, we use a pretrained 5 layer VGG-Face network [117]. This choice originates from the fact that we will be

[3] All of these trackers are trained in the way described in Section 4.3

Table 5.2: Architectural details of all the test networks.

Network	**Layers used from FEN-p**	**Layers used from FEN-c**	**Layers in the RN**
$C^{5}_{no\ att}/cp - C^{0} - F^{4}$	C5	C5	4 FC
$C^{5}_{no\ att}/cp - C^{1} - F^{3}$	C5	C5	1 C, 3 FC
$C^{3,5}_{no\ att}/cp - C^{1} - F^{3}$	C3, C5	C3, C5	1 C, 3 FC
$C^{1,2,3,4,5}_{no\ att}/cp - C^{1} - F^{3}$	C1, C2, C3, C4, C5	C1, C2, C3, C4, C5	1 C, 3 FC
$C^{1,2,3,4,5}_{att}/cp - C^{1} - F^{3}$	C1, C2, C3, C4, C5	C1, C2, C3, C4, C5	1 C, 3 FC
$C^{5}_{no\ att}/c - C^{0} - F^{4}$	-	C5	4 FC
$C^{5}_{no\ att}/c - C^{1} - F^{3}$	-	C5	1 C, 3 FC
$C^{3,5}_{no\ att}/c - C^{1} - F^{3}$	-	C3, C5	1 C, 3 FC
$C^{1,2,3,4,5}_{no\ att}/c - C^{1} - F^{3}$	-	C1, C2, C3, C4, C5	1 C, 3 FC
$C^{1,2,3,4,5}_{att}/c - C^{1} - F^{3}$	-	C1, C2, C3, C4, C5	1 C, 3 FC

tracking human faces rather than generic objects. In the RN, we use the same 4 layer MLP; however, we use 3 neurons in the last layer as our bounding boxes are in square form. Using the naming convention described above, we use $C^5_{no\ att}/cp - C^0 - F^4$ to refer to this network and use it as our baseline for comparison. The detailed block diagram of this network can be found in Figure A.1.

Adding a Fusion Network to the RN: The next test network is similar to our baseline network; however, it has a convolutional layer in place of the first fully connected layer in the RN. We use this network to test whether if exploiting the spatial information in the last layer features is helpful in regressing the bounding boxes. More specifically, the RN is composed of 1 convolutional layer followed by 3 fully connected layers where each layer has a ReLU activation. Because of this, rather than flattening and concatenating the last layer features, we concatenate them in the channel dimension with a feature adaptation network (FAN). The convolutional layer is used as a fusion network with 256 1×1 kernels, and it has a stride of 1 and a zero-padding of 0. As for the regularization, we use a batch normalization layer in the convolutional part and again a 0.5 dropout in the fully connected part. We use $C^5_{no\ att}/cp - C^1 - F^3$ to refer to this network and its block diagram is as in Figure A.2.

Using the Lower Level Features: Although the features extracted from the last layers contain rich semantic information about the target, they have a large receptive field and thus do not carry precise location information. On the other hand, the lower level features have relatively smaller receptive fields and can be useful for more precise tracking. Thus, in order to investigate the effect of the lower level features in the tracking process, we use the $C^{3,5}_{no\ att}/cp - C^1 - F^3$ network that makes use of the third (low) and fifth (high) level features. Using the FAN, the third level features are first max-pooled to match the size of the fifth level ones and then all of them are concatenated in the channel dimension. After the concatenation, they are fed as input to the RN, that is identical to the RN of $C^5_{no\ att}/cp - C^1 - F^3$, for regressing the bounding box annotation. The block diagram of this network is as in Figure A.3.

Using All Level Features: One problem that arises when using the low and high level features together is the problem of deciding which layer combination of the features to use together. Depending on the target, different layers may have different advantages and it is not immediately clear which combination will give the best tracker. Because of this, we basically use all the features and propose the $C^{1,2,3,4,5}_{no\,att}/cp - C^1 - F^3$ network whose architecture is as in Figure A.4. By this way, we leave the decision process to the fusion network inside the RN. In this network, the lower level features are first max-pooled to match the size of the last level ones and then all of them are concatenated in the channel dimension using the FAN. Finally, these concatenated features are passed through the RN, that is identical to the previous RNs with a fusion network, for the regression of the bounding box.

Using the Attention Mechanism: Although, using features from all levels provides a solution to the problem of deciding which level features to use, some of the features in these layers may be unnecessary or even harmful, whereas some may be very useful. In order to decrease or increase the effects of these channels, we propose to use a soft channel attention mechanism that explicitly weights the features before their concatenation in the FAN. More specifically, we first pass the features through a 2 layer MLP[4] to obtain the channel weights and then multiply each of the channels with each of these weights. It should be noted that the MLPs share weights across the channels of the same layer, i.e., there are 5 different MLPs that account for the 5 convolutional layers. After the concatenation in the FAN, the features are passed as usual to the RN that is again identical to the previous fusion RNs. We refer to this network as $C^{1,2,3,4,5}_{att}/cp - C^1 - F^3$ and its block diagram can be found in Figure A.5. It should be noted that this network is the proposed Attentive Face Tracking Network (AFTN). (see Section 4.1)

Using Only the Current Frames as Input: Lastly, in order to test the necessity of using the previous frames as input, we propose the versions of the above test networks that do not make use of the previous frames, i.e., there is no FEN-p. By this way, the networks will work as local face detectors and they will be able to run at speeds beyond the above trackers. We refer to these networks with the same naming conventions as the previous trackers; however, we use c's in place of the cp's.

[4] The input to this MLP is a vector of length 36 (6×6), and it has 36 neurons in its first layer and a single neuron in its last one. While the first layer has ReLU activations, the last layer has a sigmoid activation with bias 0.5. Finally, no regularization layer is used.

5.4 Testing Procedure

In order to test the trackers, we use the baseline verification protocol of the ChokePoint dataset [23]. Specifically, we first use the G1 set as our training set and train our test networks on it. Then we test the networks on the 216 video sequences of the G2 set and take the average of the evaluation metrics. For the accuracy measure, we use the area under the curve of the True Positive (TP) vs. Region Overlap Threshold (ROT) plot and for the robustness measure, we use the area above the curve of the Failure Rate (FR) vs. Reinitialization Threshold (RT) plot. It should be noted that 40 different ROT and RT values were used in drawing these two plots. After testing the trackers on the G2 set, we reset their parameters and retrain them using the G2 set. After the training is done, we test the trackers on the 216 video sequences of the G1 set and again take the average of the evaluation metrics. In the end, we report the average of the evaluation metrics on both the G1 and G2 sets as our final results.

For the speed tests, we use the average FPS values of the trackers on both the G1 and G2 sets. In detail, a tracker's speed is tested with 40 different RTs for each of the 432 different video sequences.

5.5 Quantitative Analysis

In this section, we show which pieces of our proposed tracker contributes to the most of our performance by using the test trackers described in Section 5.3. We first start by analyzing the effect of using a fusion network in the regression network and then move on to the effect of the low level features in tracking. Afterwards, rather than deciding on which combination of features to use, we look at the effect of using all them together. And then, we investigate the effect of using the channel attention mechanism. Finally, we test the necessity of using the previous frames as input. In the end, we compare all of the proposed test trackers using an Accuracy vs. Robustness plot.

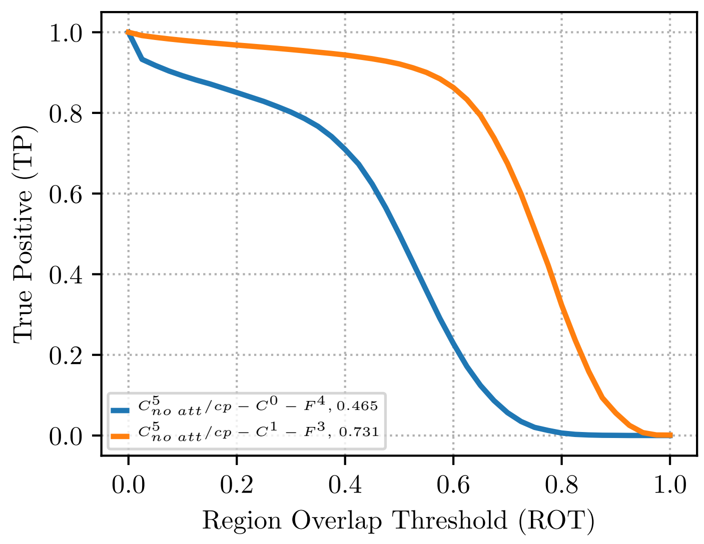


Figure 5.9: The TP vs. ROT plot for the $C^5_{no\ att}/cp - C^0 - F^4$ and $C^5_{no\ att}/cp - C^1 - F^3$ trackers. The accuracy value of the trackers is given next to their name in the legend.

5.5.1 Effect of a Fusion Network in the Regression Network

In order to test the effect of using a fusion network inside the regression network (RN), we compare the baseline tracker $C^5_{no\ att}/cp - C^0 - F^4$ with the $C^5_{no\ att}/cp - C^1 - F^3$ tracker. The only difference between these two networks is the convolutional layer that is used in place of the fully connected layer in the latter one. The performance plots of these two trackers are as in Figure 5.9 and 5.10.

The TP vs. ROT and FR vs. RT plots indicate that using a fusion network in the RN boosts the performance significantly both in terms of accuracy and robustness. This supports our hypothesis that the last level features contain spatial information and it can be exploited with a network that has convolutional layers. Thus, in the following subsections, we keep using the fusion network and build on top of this idea.

5.5.2 Effect of the Low Level Features

In order to test the effect of the lower level features, we compare the $C^5_{no\ att}/cp - C^1 - F^3$ tracker with the $C^{3,5}_{no\ att}/cp - C^1 - F^3$ tracker. The only difference between these two trackers is the additional usage of third (low) level features. The performance

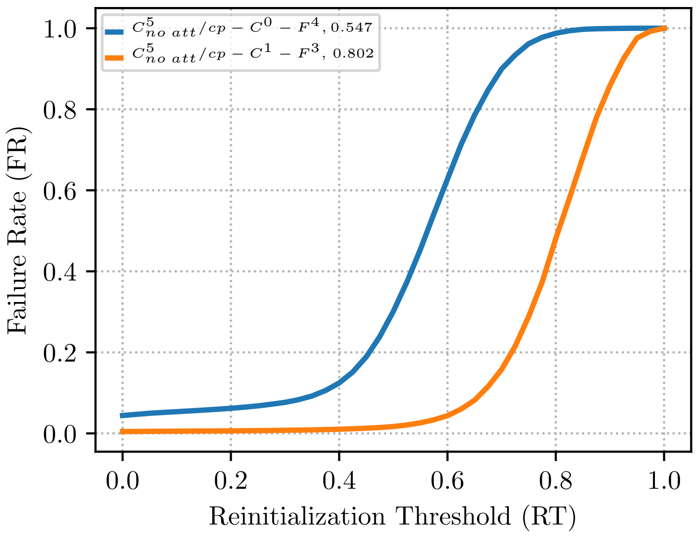


Figure 5.10: The FR vs. RT plot for the $C^5_{no\ att}/cp - C^0 - F^4$ and $C^5_{no\ att}/cp - C^1 - F^3$ trackers. The robustness value of the trackers is given next to their name in the legend.

plots of them are given in Figure 5.11 and 5.12.

The performance plots show that although the usage of the low levels features does not have an impact on the robustness, it can help with the accuracy. This aligns with our hypothesis that due to their small receptive fields, low level features can be helpful for more precise/accurate tracking.

It should be noted that although we have used the third level features as the low level features, it is not immediately clear that this is the best choice. For instance, using the first or second level features may have worked better in different tracking scenarios. In order to get rid of this manual design choice, in the next subsection, we simply use all the features from all the levels and let the fusion network decide on which level of information to use.

5.5.3 Effect of All Level Features

In order to examine the effect of using all the features, we compare the $C^{3,5}_{no\ att}/cp - C^1 - F^3$ tracker with the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ tracker. The only difference between the two is that the latter network makes use of all of the features rather than just the

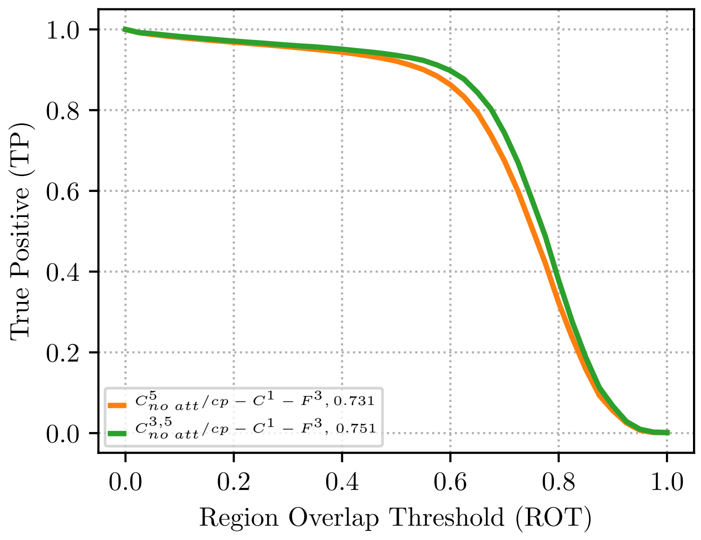


Figure 5.11: The TP vs. ROT plot for the $C^{5}_{no\ att}/cp{-}C^{1}{-}F^{3}$ and $C^{3,5}_{no\ att}/cp{-}C^{1}{-}F^{3}$ trackers. The accuracy value of the trackers is given next to their name in the legend.

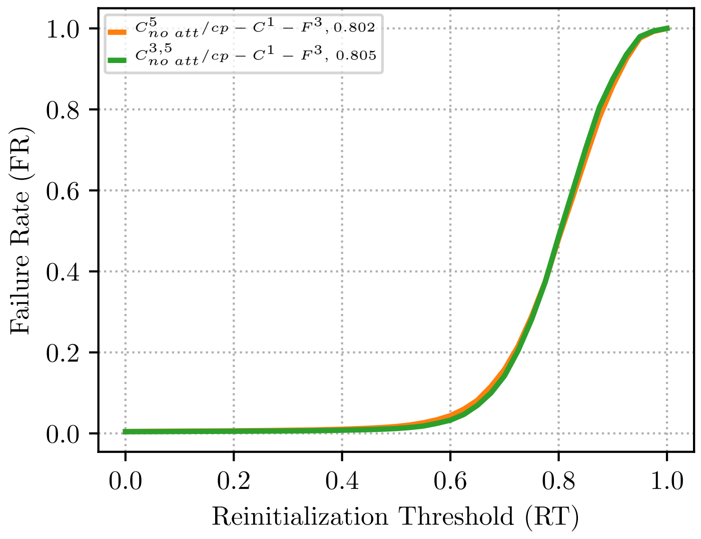


Figure 5.12: The FR vs. RT plot for the $C^{5}_{no\ att}/cp{-}C^{1}{-}F^{3}$ and $C^{3,5}_{no\ att}/cp{-}C^{1}{-}F^{3}$ trackers. The robustness value of the trackers is given next to their name in the legend.

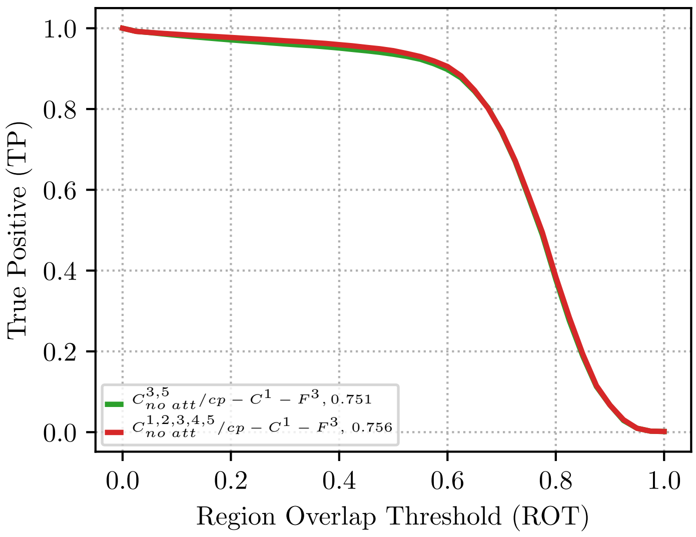


Figure 5.13: The TP vs. ROT plot for the $C^{3,5}_{no\ att}/cp - C^1 - F^3$ and $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ trackers. The accuracy value of the trackers is given next to their name in the legend.

third and fifth one. Figure 5.13 and 5.14 show the performance plots of these two trackers.

The performance plots show that although the accuracy of the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ tracker is slightly better, they have a very close accuracy and robustness profile. This is because, in this specific face tracking scenario, the third and fifth level features already contain enough information for accurate/robust tracking and adding the other level features does not help. However, using all the features takes away the burden of testing all the possible combinations and leaves this job to the fusion network.

Although the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ tracker takes away the testing burden, it makes use of all the features from all the layers regardless of their importance in tracking. Even though the fusion network takes care of the fusion of these channels, using an explicit mechanism for gating these channels can lead to better results. In the next subsection, we investigate the effect of such an explicit gating mechanism.

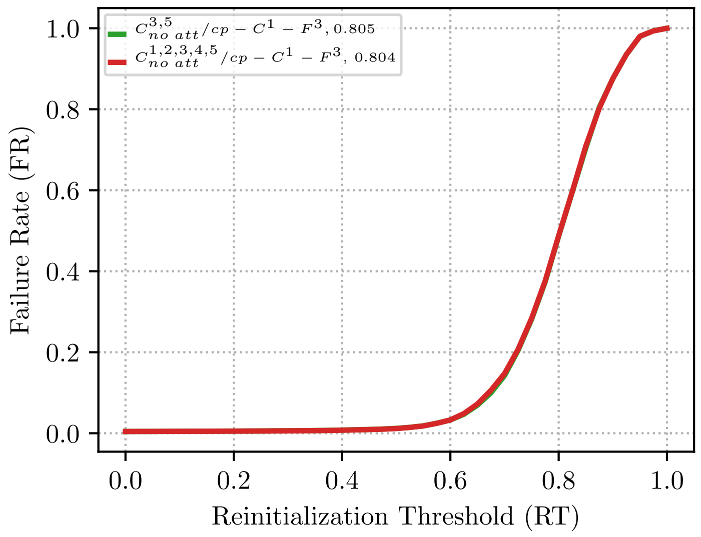


Figure 5.14: The FR vs. RT plot for the $C^{3,5}_{no\ att}/cp{-}C^1{-}F^3$ and $C^{1,2,3,4,5}_{no\ att}/cp{-}C^1{-}F^3$ trackers. The robustness value of the trackers is given next to their name in the legend.

5.5.4 Effect of the Channel Attention Mechanism

Lastly, in order to test the effect of the soft channel attention mechanism, we compare the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ tracker with the $C^{1,2,3,4,5}_{att}/cp - C^1 - F^3$ tracker. The only difference between these two trackers is the channel attention mechanism present in the latter one. The performance plots of these two trackers are given in Figure 5.15 and 5.16.

The performance plots indicate that the usage of an explicit gating mechanism as the soft channel attention mechanism brings a performance gain both in terms of accuracy and robustness. This assists our hypothesis that certain channels may increase/decrease the tracking performance and explicitly weighting these channels can be helpful in obtaining more accurate/robust trackers.

5.5.5 Effect of Using Only the Current Frame

In this subsection, we test the necessity of using the previous frames as input. In order to do this, we compare the versions of the trackers that only use the current frames as input with themselves. In this case, the trackers will work as local face detectors. The

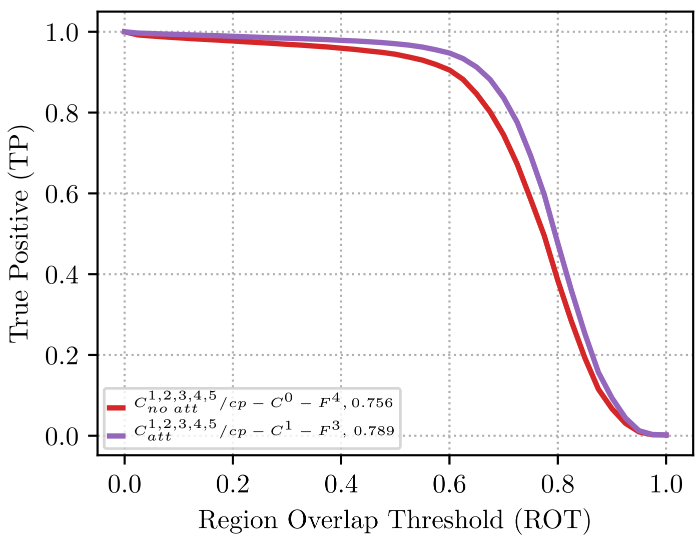


Figure 5.15: The TP vs. ROT plot for the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ and $C^{1,2,3,4,5}_{att}/cp - C^1 - F^3$ trackers. The accuracy value of the trackers is given next to their name in the legend.

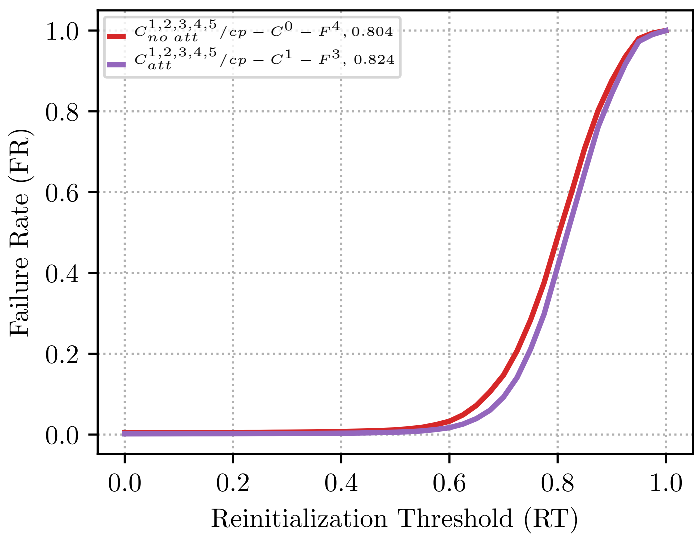


Figure 5.16: The FR vs. RT plot for the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ and $C^{1,2,3,4,5}_{att}/cp - C^1 - F^3$ trackers. The robustness value of the trackers is given next to their name in the legend.

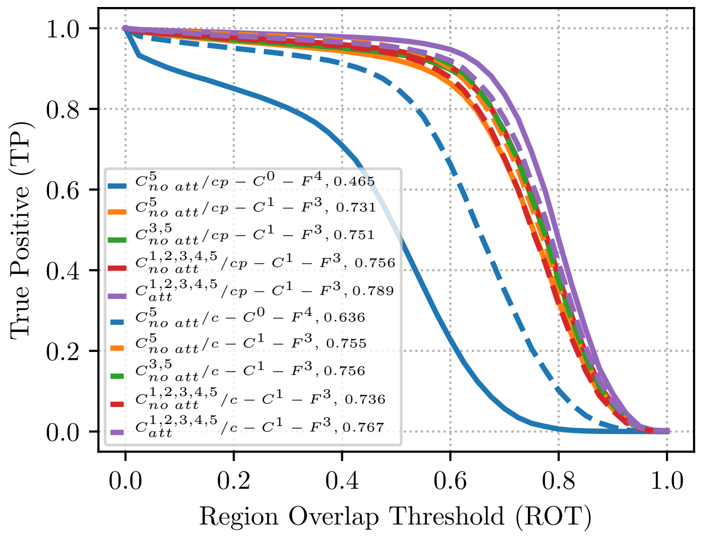


Figure 5.17: The TP vs. ROT plot for the all of the test trackers. The accuracy value of the trackers is given next to their name in the legend. The trackers that do not use FEN-p are represented with dashed lines.

performance plots of these trackers are as in Figure 5.17 and 5.18.

The performance plots show that although there are slight variations, a similar trend of increasing performance with the addition of useful pieces also holds for trackers that do not use FEN-p. These trackers also perform closely to the trackers that use both of the frames as their inputs.[5] This raises questions on the necessity of using the previous frames, which slows down the trackers unnecessarily. Whether they should be used or not will be discussed further in Section 5.7.

5.5.6 The Overall Comparison of the Trackers

After the one-by-one comparison of the test trackers, in this subsection, we compare all of them using a single Accuracy (A) vs. Robustness (R) plot given in Figure 5.19. The individual accuracy and robustness values are also presented in Table 5.3. As it is clear from the plot and the table, the best performing trackers in terms of both accuracy and robustness are the $C^{1,2,3,4,5}_{att}/cp - C^{1} - F^{3}$ (AFTN) and $C^{1,2,3,4,5}_{att}/c -$

[5] The huge performance increase in the baseline tracker ($C^{5}_{no\ att}/cp - C^{0} - F^{4}$) is due to the prevention of overfitting which is caused by the large number of parameters in its RN. The large number of parameters is due to the first fully connected layer in the RN.

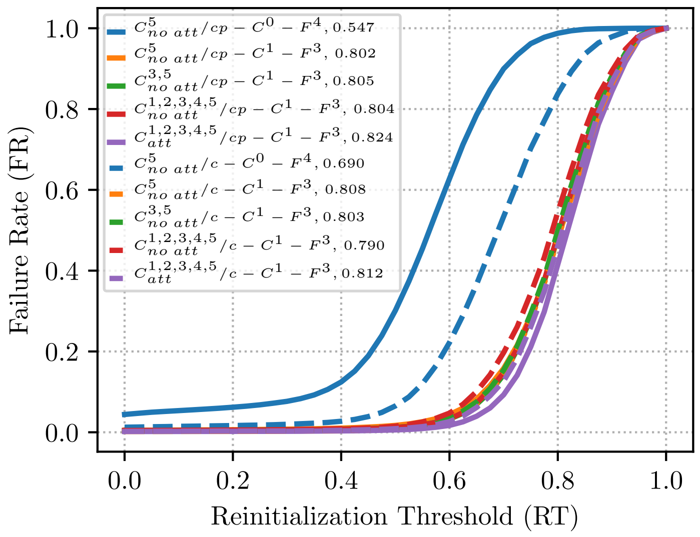


Figure 5.18: The FR vs. RT plot for the all of the test trackers. The accuracy value of the trackers is given next to their name in the legend. The trackers that do not use FEN-p are represented with dashed lines.

$C^1 - F^3$ (AFTN-c) trackers. This is as expected, as these trackers explicitly weight the channels that are necessary/dispensable for more accurate and robust tracking. Another interesting thing to observe is the significant effect that the fusion network brings. As can be seen from the A vs. R plot, it makes the trackers cluster near to the top-right corner.

5.6 Speed Analysis

In this section, we compare the speeds of the test trackers that were described above. In order to make a fair comparison, we run all the trackers on a machine equipped with an Intel Core i7-4790K 8 Core 4.00 GHz CPU and a single NVIDIA GeForce GTX Titan X GPU. We also use the PyTorch framework [46] and enable its benchmark mode during the tracking process.

The tracking speeds of all the test trackers are as in Table 5.4. These speed values were calculated by using the time it takes for cropping, resizing and forward passing the input frames. It should be noted that in our calculations, we do not take into

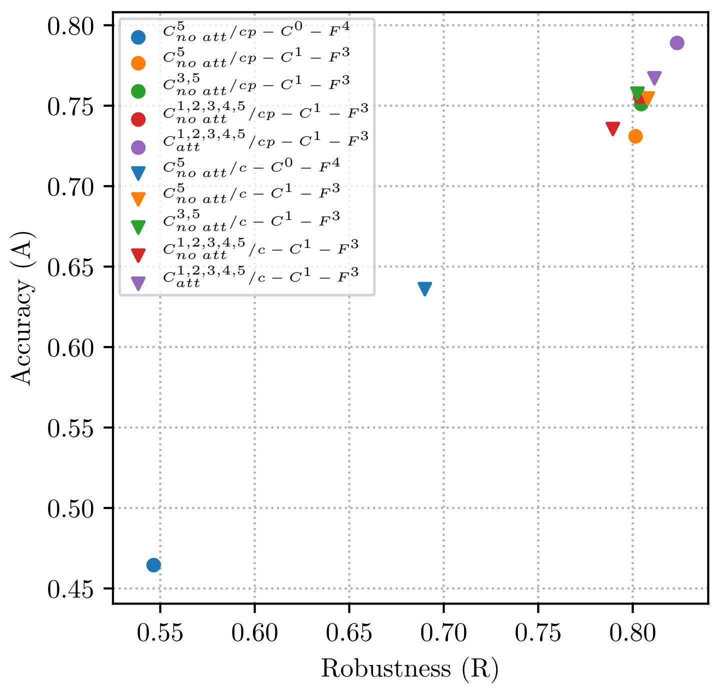


Figure 5.19: The A vs. R plot for all of the test trackers.

Table 5.3: Accuracy and robustness values of all the test trackers. The overall score is obtained by averaging the accuracy and robustness values.

Network	**Accuracy**	**Robustness**	**Overall**
$C^5_{no\ att}/cp - C^0 - F^4$	0.465	0.547	0.506
$C^5_{no\ att}/cp - C^1 - F^3$	0.731	0.802	0.767
$C^{3,5}_{no\ att}/cp - C^1 - F^3$	0.751	0.805	0.778
$C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$	0.756	0.804	0.780
$C^{1,2,3,4,5}_{att}/cp - C^1 - F^3$	**0.789**	**0.824**	**0.807**
$C^5_{no\ att}/c - C^0 - F^4$	0.636	0.690	0.663
$C^5_{no\ att}/c - C^1 - F^3$	0.755	0.808	0.782
$C^{3,5}_{no\ att}/c - C^1 - F^3$	0.756	0.803	0.780
$C^{1,2,3,4,5}_{no\ att}/c - C^1 - F^3$	0.736	0.790	0.763
$C^{1,2,3,4,5}_{att}/c - C^1 - F^3$	**0.767**	**0.812**	**0.790**

Table 5.4: Speed values of all the test trackers.

Network	**Speed (FPS)**
$C^{5}_{no\ att}/cp - C^{0} - F^{4}$	118.6
$C^{5}_{no\ att}/cp - C^{1} - F^{3}$	156.8
$C^{3,5}_{no\ att}/cp - C^{1} - F^{3}$	148.3
$C^{1,2,3,4,5}_{no\ att}/cp - C^{1} - F^{3}$	148.9
$C^{1,2,3,4,5}_{att}/cp - C^{1} - F^{3}$	**142.9**
$C^{5}_{no\ att}/c - C^{0} - F^{4}$	182.5
$C^{5}_{no\ att}/c - C^{1} - F^{3}$	207.9
$C^{3,5}_{no\ att}/c - C^{1} - F^{3}$	206.4
$C^{1,2,3,4,5}_{no\ att}/c - C^{1} - F^{3}$	199.2
$C^{1,2,3,4,5}_{att}/c - C^{1} - F^{3}$	**183.4**

account the time it takes for OpenCV to read a frame. The FPS values in Table 5.4 indicate that the extensions do not have a significant computational overhead and in fact they even speed up the baseline tracker. They also show that trackers that only use the current frames perform much faster than the other ones.

Lastly, one important thing to note is that all of the test trackers can run at speeds that are very far beyond the 25 FPS requirement for real-time tracking. This is mainly due to the following two aspects: the trackers are trained fully offline with no online updating involved and only a single forward pass is required for inferring the bounding box annotations. The usage of a GPU, rather than a CPU, is another important aspect that significantly contributes to these results.

5.7 Which Tracker to Use?

In the previous sections, we have demonstrated the usefulness of several extensions through quantitative accuracy and robustness analysis, and showed that the best performing trackers are the $C^{1,2,3,4,5}_{att}/cp - C^{1} - F^{3}$ (AFTN) and $C^{1,2,3,4,5}_{att}/c - C^{1} - F^{3}$

(AFTN-c) trackers. Among these two, the former one has a higher overall score whereas the latter performs much faster. So which one of these two trackers should be used for face tracking?

The answer to this question is partially available in the study of Held et al. [14] in which they compare the GOTURN tracker with its version where no previous frame is used. They show that the tracker which receives both the current and previous frames performs better in the overall score when there is an occlusion or a large camera motion, and the tracker which receives just the current frame performs better when there is a large object size change. Since there are no severe occlusions and camera motions in the ChokePoint dataset [23], using the AFTN-c tracker also does not cause serious problems in the overall score. Moreover, it has the advantage in terms of speed. Thus, the AFTN-c tracker, which acts as a local face detector, serves well under these conditions and it can be used for tracking faces. However, if the overall score is an important concern, then the AFTN tracker can be used instead. So the choice of which tracker to use depends on the objective (speed or overall score) that we want to maximize.

It should also be noted that if face tracking is to be performed under different conditions than the ChokePoint dataset, then the choice of whether or not using the previous frame should be investigated further. However, using both of the frames may be a better choice if speed is not much of a concern as modern surveillance cameras can record in speeds that prevent large changes in the object size.

5.8 Qualitative Analysis: Learning to Select Useful Features

The previous sections have demonstrated the usefulness of the channel-wise attention mechanism through a quantitative analysis. In this section, we provide some additional visualizations to demonstrate its usefulness in a qualitative manner.

Figure 5.21 visualizes the average channel weights for all the convolutional layers of the same person with ID 18 (see Figure 5.20) in the video sequences P1E_S4_C1 and P1L_S4_C1. Since the last layer of the attention network has a sigmoid with bias 0.5, the weights are in the range $[0.5, 1.5]$. First, we observe that the weight

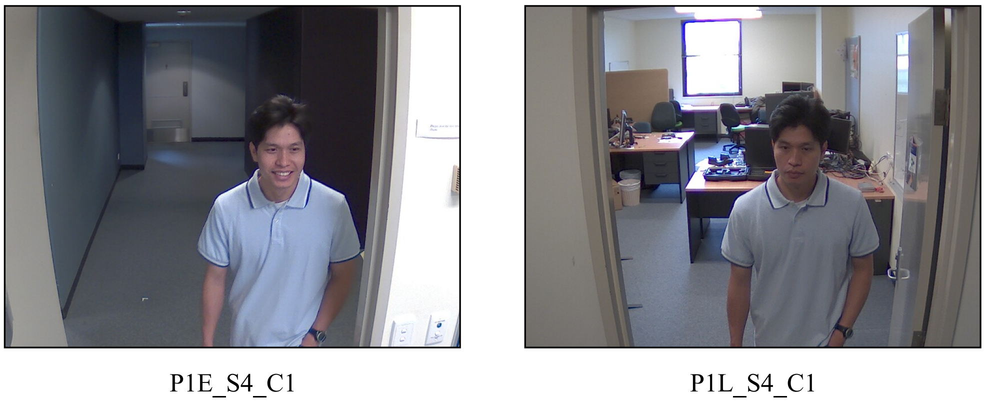


Figure 5.20: Entering and leaving scenarios of person 18 from P1. The leaving scenario (P1L) contains more distracting objects in the background.

distributions of the layers are quite different, i.e., while some of the first level features get suppressed by the attention mechanism, the features from rest of the layers are passed without nearly no weighting. This is as expected as the lower layer features contain more generic information that may not be necessary for the tracking process. Second, the weight distributions for the first layer C1 are different for the two video sequences. The attention mechanism suppresses more channels from C1 in the P1L sequence as it contains more distracting objects in the background.

Lastly, it should be noted that the weights in Figure 5.21 are for the AFTN tracker. However, the weights for the AFTN-c tracker also show a similar distribution where they again supress the first layer features more than the other ones. They also supress more features from the first layer of the P1L sequence.

5.9 Comparison with Other Surveillance Face Trackers

The previous sections were about building on top of the GOTURN tracker [14] to obtain the AFTN tracker and its single-input-frame version AFTN-c. In this section, we compare our proposed tracker to the other video surveillance trackers, that were presented in Section 3.3, to observe our place among them.[6] Among the trackers that

[6] We compare the trackers only in terms of accuracy and robustness as it would be unfair to compare the speeds of trackers that run on powerful GPUs with the ones that run on only CPUs.

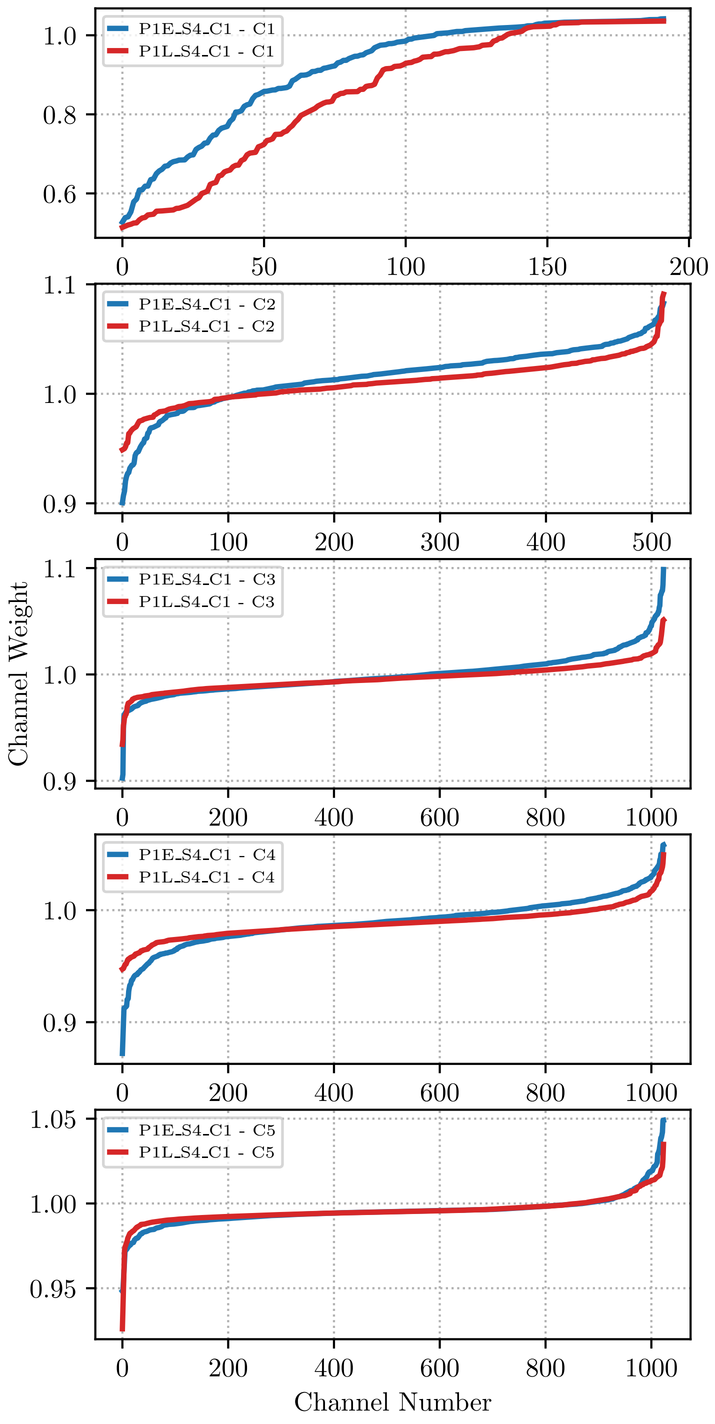


Figure 5.21: The average channel weight outputs of the AFTN network by the attention mechanism for the same person with ID 18 in the video sequences P1E_S4_C1 and P1L_S4_C1. Channels are sorted according to their weights. There is no correspondence between channel numbers for the two sequences.

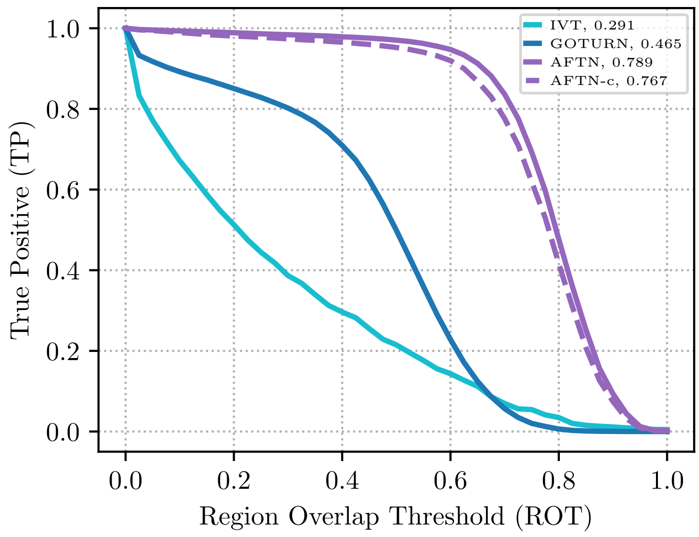


Figure 5.22: The TP vs. ROT plot for the IVT, GOTURN, AFTN and AFTN-c trackers. The accuracy value of the trackers is given next to their name in the legend.

were used by Dewan et al. [1], we only use the IVT (Incremental Visual Tracking) [22] tracker for comparison as they have already shown that it performs better than both TLD [89] and DSCT [90] in terms of tracking accuracy and speed. It should be noted that we force the IVT tracker to output squares rather than arbitrary polygons to make a fair comparison with our trackers. We were not able to use the HAKLT tracker [2] for comparison as its source code is not publicly available. The performance plots of the IVT, GOTURN, AFTN and AFTN-c trackers are as in Figure 5.22 and 5.23.

The performance plots show that the IVT tracker performs even worse than our baseline GOTURN tracker. This assists our hypothesis that although traditional trackers can perform up to certain degrees, since they do not make use of the learned rich hierarchical features present in deep learning based trackers, they are prone to failure under the harsh conditions present in video surveillance.

Lastly, it should be noted that the experiments with IVT tracker were done with optimized hyperparameters that are close to the the default hyperparameters in the original paper [22]. Specifically, for the eigenbasis representation, each target region is resized to 32×32 and 16 eigenvectors are used. The forgetting term is set to 0.99, the batch size for the eigenbasis update is set to 10 and 300 particles are used.

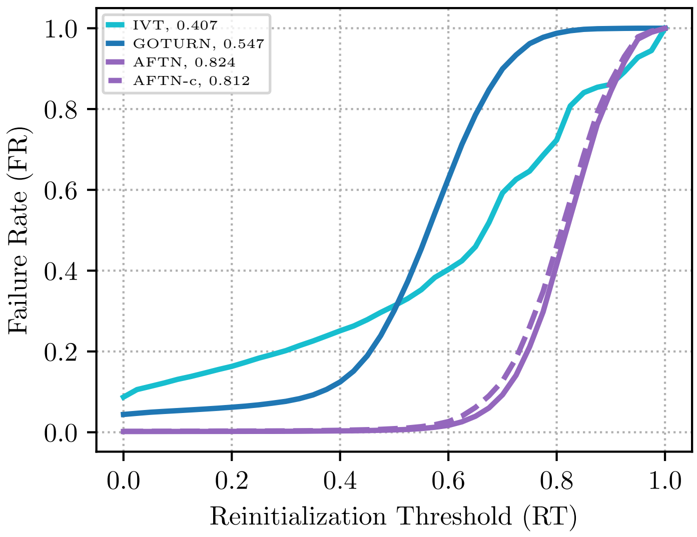


Figure 5.23: The FR vs. RT plot for the IVT, GOTURN, AFTN and AFTN-c trackers. The accuracy value of the trackers is given next to their name in the legend.

CHAPTER 6

CONCLUSION

The aim of this thesis was to design a real-time visual object tracker for the specific problem of single-target visual face tracking under surveillance conditions. For this purpose, we took the real-time GOTURN tracker [14] that makes use of the learned rich hierarchical features as our starting point and improved it using several useful extensions as using a fusion network in the regression network, making use of the lower level features and using a channel-wise attention mechanism for the task of adaptive channel selection. We demonstrated the usefulness of all these extensions using careful experiments and finally proposed the Attentive Face Tracking Network (AFTN). We also ran experiments to check the necessity of using the previous frames as input and showed that it may not be necessary if tracker speed is a concern. As shown using an Accuracy vs. Robustness plot, the AFTN and its single-frame-input version AFTN-c outperform all the other trackers that are the primitive versions of themselves. They also outperform one of the best [1] surveillance face tracker named Incremental Visual Tracking (IVT) [22] by a very large margin. Furthermore, they run at speeds ($\sim$140 FPS or $\sim$180 FPS) that are very far beyond the requirement of 25 FPS for real-time tracking making them useful for real-world applications.

It should be noted that although we specifically used the proposed AFTN (and AFTN-c) for the task of real-time single-target visual face tracking, it can also be used for any real-time single-target visual generic object tracking task without any further modification in the architecture. It just needs to be re-trained from scratch in an offline manner with the domain specific dataset.

In addition to proposing a real-time face tracker, this thesis also provided accurate bounding box annotations for the G1 and G2 sets of the ChokePoint dataset [23] which can be used for further studies in visual face tracking under surveillance conditions. The variance of the illumination conditions, pose, sharpness and misalignment of the face images in this dataset can serve well in training trackers that are robust to the wide variety of changes in the environment. Our results on the Accuracy vs. Robustness plot are already a strong indicator of this benefit.

It should also be noted that the ChokePoint dataset consists of sequences that have a single face in each frame. As we were building on top of a single-target object tracker for tracking in this dataset, this did not cause a problem in our experiments. However, if the task is to track multiple faces, then AFTN (or AFTN-c) may not be a good choice as running a separate tracker, on a machine with single CPU and GPU, for each of the faces may become computationally expensive and slow down the tracker. In this case, multi-target object trackers can be investigated further.

The proposed face tracking architecture opens many possible future directions that can be further investigated. The first obvious one is the effect of a hard channel attention mechanism that does not weights all the channels as in the soft case, but selects a certain number of useful channels that can help in the process of tracking. Another possible direction is the investigation of the effect of adding a recurrent module to the regression network to make use of the spatio-temporal information in a different way. In this case, there would also be no need for using the two consecutive frames as input which in turn would lead to a speed up.

Apart from the possible future studies to improve the tracker, we also plan to augment our face tracker using a face detection and a face recognition module to build a large framework that automatically detects-tracks-recognizes the faces under real-world surveillance conditions. After the faces are detected with the detection module, they will be tracked by the tracking module and the identity of them will be determined using the recognition module. In the case of a tracking failure, the detection module will re-detect the faces and the whole process will continue from where it was left.

Before ending this thesis, we highlight the need for empirical benchmarking studies in surveillance face tracking as the Visual Object Tracking (VOT) challenges [18–21]. By this way, the surveillance face tracking community would be able to easily compare their face tracking methods with the ones in the literature and thus the field would progress faster.

REFERENCES

[1] M. A. A. Dewan, E. Granger, F. Roli, R. Sabourin, and G. L. Marcialis, "A comparison of adaptive appearance methods for tracking faces in video surveillance," in *5th International Conference on Imaging for Crime Detection and Prevention (ICDP 2013)*, pp. 1–7, Dec 2013.

[2] X. Lan, Z. Xiong, W. Zhang, S. Li, H. Chang, and W. Zeng, "A super-fast online face tracking system for video surveillance," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1998–2001, May 2016.

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *CoRR*, vol. abs/1405.3531, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, p. 2012.

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

[6] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014.

[7] S. Hong, H. Noh, and B. Han, "Decoupled deep neural network for semi-supervised semantic segmentation," in *NIPS*, 2015.

[8] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, pp. 640–651, Apr. 2017.

[9] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic

segmentation," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[10] H. Noh, P. H. Seo, and B. Han, "Image question answering using convolutional neural network with dynamic parameter prediction," *CoRR*, vol. abs/1511.05756, 2015.

[11] Y. Taigman, M. Yang, and L. Wolf, "L.: Deepface: Closing the gap to human-level performance in face verification," in *In: IEEE CVPR*, 2014.

[12] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1653–1660, 2014.

[13] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference Computer Vision (ECCV)*, 2016.

[15] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," *arXiv preprint arXiv:1606.09549*, 2016.

[16] H. Nam, M. Baek, and B. Han, "Modeling and propagating cnns in a tree structure for visual tracking," *CoRR*, vol. abs/1608.07242, 2016.

[17] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8971–8980, 2018.

[18] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Čehovin Zajc, G. Fernandez, T. Vojir, G. Häger, G. Nebehay, R. Pflugfelder, A. Gupta, A. Bibi, A. Lukežič, A. Garcia-Martin, A. Saffari, A. Petrosino, A. S. Montero, A. Varfolomieiev, A. Baskurt, B. Zhao, B. Ghanem, B. Martinez, B. Lee, B. Han, C. Wang, C. Garcia, C. Zhang, C. Schmid, D. Tao, D. Kim, D. Huang,

D. Prokhorov, D. Du, D.-Y. Yeung, E. Ribeiro, F. S. Khan, F. Porikli, F. Bunyak, G. Zhu, G. Seetharaman, H. Kieritz, H. T. Yau, H. Li, H. Qi, H. Bischof, H. Possegger, H. Lee, H. Nam, I. Bogun, J. chan Jeong, J. il Cho, J.-Y. Lee, J. Zhu, J. Shi, J. Li, J. Jia, J. Feng, J. Gao, J. Y. Choi, J.-W. Kim, J. Lang, J. M. Martinez, J. Choi, J. Xing, K. Xue, K. Palaniappan, K. Lebeda, K. Alahari, K. Gao, K. Yun, K. H. Wong, L. Luo, L. Ma, L. Ke, L. Wen, L. Bertinetto, M. Pootschi, M. Maresca, M. Danelljan, M. Wen, M. Zhang, M. Arens, M. Valstar, M. Tang, M.-C. Chang, M. H. Khan, N. Fan, N. Wang, O. Miksik, P. Torr, Q. Wang, R. Martin-Nieto, R. Pelapur, R. Bowden, R. Laganiere, S. Moujtahid, S. Hare, S. Hadfield, S. Lyu, S. Li, S.-C. Zhu, S. Becker, S. Duffner, S. L. Hicks, S. Golodetz, S. Choi, T. Wu, T. Mauthner, T. Pridmore, W. Hu, W. Hübner, X. Wang, X. Li, X. Shi, X. Zhao, X. Mei, Y. Shizeng, Y. Hua, Y. Li, Y. Lu, Y. Li, Z. Chen, Z. Huang, Z. Chen, Z. Zhang, and Z. He, "The visual object tracking vot2015 challenge results," in *Visual Object Tracking Workshop 2015 at ICCV2015*, Dec 2015.

[19] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. Čehovin Zajc, T. Vojir, G. Häger, A. Lukežič, and G. Fernandez, "The visual object tracking vot2016 challenge results." Springer, Oct 2016.

[20] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. Čehovin Zajc, T. Vojir, G. Häger, A. Lukežič, A. Eldesokey, and G. Fernandez, "The visual object tracking vot2017 challenge results," 2017.

[21] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pfugfelder, L. C. Zajc, T. Vojir, G. Bhat, A. Lukezic, A. Eldesokey, G. Fernandez, and et al., "The sixth visual object tracking vot2018 challenge results," 2018.

[22] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *Int. J. Comput. Vision*, vol. 77, pp. 125–141, May 2008.

[23] Y. Wong, S. Chen, S. Mau, C. Sanderson, and B. C. Lovell, "Patch-based probabilistic image quality assessment for face selection and improved video-based face recognition," in *IEEE Biometrics Workshop, Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 81–88, IEEE, June 2011.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, November 1998.

[25] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

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

[27] A. M. Legendre, *Nouvelles méthodes pour la détermination des orbites des cometes*. F. Didot, 1805.

[28] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. 1809.

[29] C. F. Gauss, *Theoria combinationis observationum erroribus minimis obnoxiae (Theory of the combination of observations least subject to error)*. 1821.

[30] W. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 7, pp. 115–133, 1943.

[31] D. O. Hebb, *The Organization of Behavior*. Wiley, New York, 1949.

[32] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[33] M. Minsky and S. Papert, *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[34] J. Schmidhuber, “Critique of paper by "deep learning conspiracy" (nature 521 p 436),” June 2015. `http://people.idsia.ch/~juergen/deep-learning-conspiracy.html`.

[35] A. G. Ivakhnenko and V. G. Lapa, *Cybernetic Predicting Devices*. CCM Information Corporation, 1965.

[36] A. G. Ivakhnenko, “The group method of data handling – a rival of the method of stochastic approximation,” *Soviet Automatic Control*, vol. 13, no. 3, pp. 43–55, 1968.

[37] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings National Academy of Science*, vol. 79, pp. 2554–2558, April 1982.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, pp. 318–362, MIT Press, 1986.

[39] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *Proc. International Conference on Computer Vision (ICCV'09)*, pp. 2146–2153, IEEE, 2009.

[40] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier networks," in *AISTATS*, vol. 15, pp. 315–323, 2011.

[41] A. N. Kolmogorov, "On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition," *Doklady Akademii. Nauk USSR,*, vol. 114, pp. 679–681, 1965.

[42] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec 1989.

[43] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[44] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, pp. 930–945, May 1993.

[45] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2924–2932, Curran Associates, Inc., 2014.

[46] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[47] H. J. Kelley, “Gradient theory of optimal flight paths,” *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[48] A. E. Bryson, “A gradient method for optimizing multi-stage allocation processes,” in *Proc. Harvard Univ. Symposium on digital computers and their applications*, 1961.

[49] S. E. Dreyfus, “The numerical solution of variational problems,” *Journal of Mathematical Analysis and Applications*, vol. 5(1), pp. 30–45, 1962.

[50] A. E. Bryson, Jr. and W. F. Denham, “A steepest-ascent method for solving optimum programming problems,” Tech. Rep. BR-1303, Raytheon Company, Missle and Space Division, 1961.

[51] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. 1961.

[52] J. H. Wilkinson, ed., *The Algebraic Eigenvalue Problem*. New York, NY, USA: Oxford University Press, Inc., 1965.

[53] S. Amari, “A theory of adaptive pattern classifiers,” *IEEE Trans. EC*, vol. 16, no. 3, pp. 299–307, 1967.

[54] A. Bryson and Y. Ho, *Applied optimal control: optimization, estimation, and control*. Blaisdell Pub. Co., 1969.

[55] S. W. Director and R. A. Rohrer, “Automated network design - the frequency-domain case,” *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 330–337, 1969.

[56] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1st ed., 1957.

[57] S. Linnainmaa, “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors,” Master’s thesis, Univ. Helsinki, 1970.

[58] S. Linnainmaa, “Taylor expansion of the accumulated rounding error,” *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.

[59] S. E. Dreyfus, “The computational solution of optimal control problems with time lag,” *IEEE Transactions on Automatic Control*, vol. 18(4), pp. 383–385, 1973.

[60] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[61] P. J. Werbos, “Applications of advances in nonlinear sensitivity analysis,” in *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pp. 762–770, 1981.

[62] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[63] J. Hadamard, *Mémoire sur le problème d’analyse relatif à l’équilibre des plaques élastiques encastrées*. Mémoires présentés par divers savants à l’Académie des sciences de l’Institut de France: Éxtrait, Imprimerie nationale, 1908.

[64] T. Tieleman and G. Hinton, “Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.” COURSERA: Neural Networks for Machine Learning, 2012.

[65] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014.

[66] T. Schaul, I. Antonoglou, and D. Silver, “Unit tests for stochastic optimization,” in *International Conference on Learning Representations*, (Banff, Canada), 2014.

[67] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *ArXiv e-prints*, mar 2016.

[68] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *J. Physiol.*, vol. 148, pp. 574–591, 1959.

[69] D. H. Hubel and T. Wiesel, “Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex,” *Journal of Physiology (London)*, vol. 160, pp. 106–154, 1962.

[70] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron," *Trans. IECE*, vol. J62-A(10), pp. 658–665, 1979.

[71] K. Fukushima, "Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[72] K. Fukushima, "Training multi-layered neural network Neocognitron," *Neural Networks*, vol. 40, pp. 18–31, 2013.

[73] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Back-propagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS 2012)*, p. 4, 2012.

[75] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[77] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[78] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[79] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," *CoRR*, vol. abs/1801.05134, 2018.

[80] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[81] R. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME - Journal of basic Engineering*, vol. 82, pp. 35–45, 01 1960.

[82] H. Kushner, “Dynamical equations for optimal nonlinear filtering,” *Journal of Differential Equations*, vol. 3, no. 2, pp. 179 – 190, 1967.

[83] P. D. Moral, “Nonlinear filtering: Interacting particle resolution,” 1996.

[84] P. Del Moral, “Measure-valued processes and interacting particle systems. application to nonlinear filtering problems,” *Ann. Appl. Probab.*, vol. 8, pp. 438–495, 05 1998.

[85] D. Casasent, “Unified synthetic discriminant function computational formulation,” *Appl. Opt.*, vol. 23, pp. 1620–1627, May 1984.

[86] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 603–619, May 2002.

[87] D. Comaniciu, V. Ramesh, and P. Meer, “Kernel-based object tracking,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 564–577, May 2003.

[88] J. Shi and C. Tomasi, “Good features to track,” in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, June 1994.

[89] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, pp. 1409–1422, July 2012.

[90] Q. Wang, Feng Chen, Wenli Xu, and M. Yang, “Online discriminative object tracking with local sparse representation,” in *2012 IEEE Workshop on the Applications of Computer Vision (WACV)*, pp. 425–432, Jan 2012.

[91] P. Li, D. K. Wang, L. Wang, and H. Lu, “Deep visual tracking: Review and experimental comparison,” *Pattern Recognition*, vol. 76, pp. 323–338, 2018.

[92] S. Krebs, B. Duraisamy, and F. Flohr, “A survey on leveraging deep neural networks for object tracking,” *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 411–418, 2017.

[93] N. Wang and D.-Y. Yeung, “Learning a deep compact image representation for visual tracking,” in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’13, (USA), pp. 809–817, Curran Associates Inc., 2013.

[94] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, “Hierarchical convolutional features for visual tracking,” in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV ’15, (Washington, DC, USA), pp. 3074–3082, IEEE Computer Society, 2015.

[95] L. Wang, W. Ouyang, X. Wang, and H. Lu, “Visual tracking with fully convolutional networks,” *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 3119–3127, 2015.

[96] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Volume 1 - Volume 01*, CVPR ’05, (Washington, DC, USA), pp. 539–546, IEEE Computer Society, 2005.

[97] R. Tao, E. Gavves, and A. W. M. Smeulders, “Siamese instance search for tracking,” *CoRR*, vol. abs/1605.05863, 2016.

[98] Q. Gan, Q. Guo, Z. Zhang, and K. Cho, “First step toward model-free, anonymous object tracking with recurrent neural networks,” *CoRR*, vol. abs/1511.06425, 2015.

[99] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[100] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once:

Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[101] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang, "Spatially supervised recurrent convolutional neural networks for visual object tracking," *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.

[102] J. Choi, H. J. Chang, J. Jeong, Y. Demiris, and J. Y. Choi, "Visual tracking using attention-modulated disintegration and integration," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4321–4330, June 2016.

[103] J. Choi, H. J. Chang, S. Yun, T. Fischer, Y. Demiris, and J. Y. Choi, "Attentional correlation filter network for adaptive visual tracking," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4828–4837, July 2017.

[104] S. E. Kahou, V. Michalski, R. Memisevic, C. J. Pal, and P. Vincent, "Ratm: Recurrent attentive tracking model," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1613–1622, 2017.

[105] A. R. Kosiorek, A. Bewley, and I. Posner, "Hierarchical attentive recurrent tracking," in *NIPS*, 2017.

[106] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[107] A. He, C. Luo, X. Tian, and W. Zeng, "A twofold siamese network for real-time object tracking," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4834–4843, 2018.

[108] Q. Wang, Z. Teng, J. Xing, J. Gao, W. Hu, and S. Maybank, "Learning attentions: Residual attentional siamese network for high performance online visual tracking," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[109] D. Chen, S. Ren, Y. Wei, X. Cao, and J. Sun, "Joint cascade face detection and alignment," in *ECCV*, 2014.

[110] Z. Ren, S. Yang, F. Zou, F. Yang, C. Luan, and K. Li, "A face tracking framework based on convolutional neural networks and kalman filter," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 410–413, Nov 2017.

[111] I. Choi and Y. Kim, "Deep manifold embedding active shape model for pose invarient face tracking," in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 578–581, Jan 2018.

[112] O. Parkhi, E. Rahtu, Q. Cao, and A. Zisserman, "Automated video face labelling for films and tv material," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2018.

[113] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, pp. 1499–1503, Oct 2016.

[114] M. Mathias, R. Benenson, M. Pedersoli, and L. V. Gool, "Face detection without bells and whistles," in *ECCV*, 2014.

[115] İbrahim Batuhan Akkaya, *Mouse Face Tracking Using Convolutional Neural Networks*. M.Sc. Thesis, Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey, 2016.

[116] I. B. Akkaya and U. Halici, "Mouse face tracking using convolutional neural networks," *IET Computer Vision*, vol. 12, no. 2, pp. 153–161, 2018.

[117] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015.

[118] "Chokepoint dataset webpage." `http://arma.sourceforge.net/chokepoint/`. Accessed: 2019-06-20.

[119] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[120] L. Čehovin Zajc, M. Kristan, and A. Leonardis, "Is my new tracker really better than yours?," in *WACV 2014: IEEE Winter Conference on Applications of Computer Vision*, IEEE, Mar 2014.

APPENDIX A

ARCHITECTURAL DETAILS OF THE TEST NETWORKS

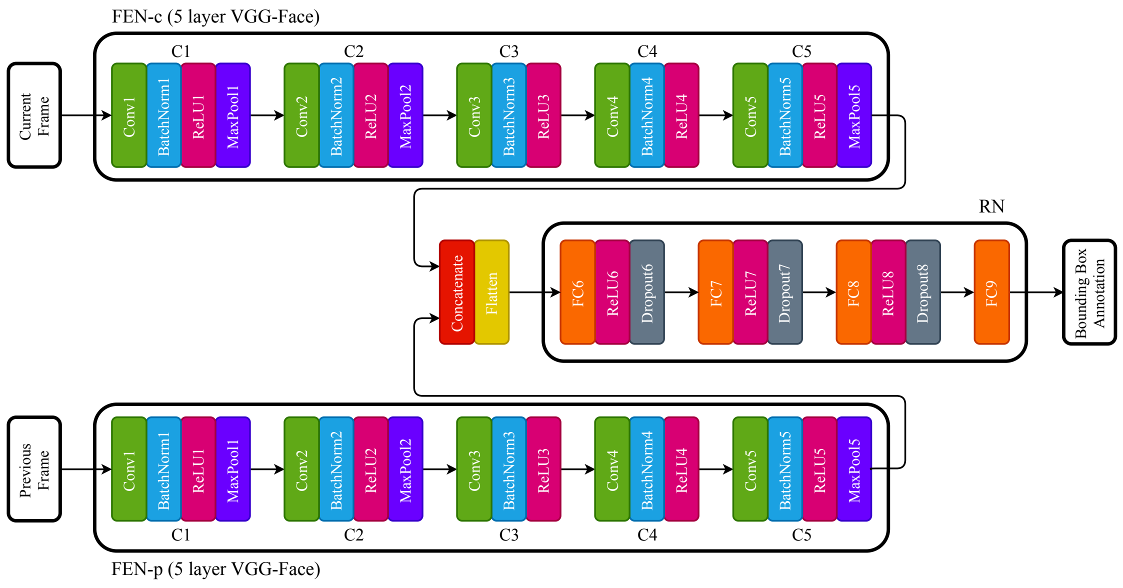


Figure A.1: The network architecture of the $C^5_{no\ att}/cp - C^0 - F^4$ tracker. It has the same architecture with the GOTURN tracker. However, a pretrained VGG-Face network is used for feature extraction rather than an AlexNet.

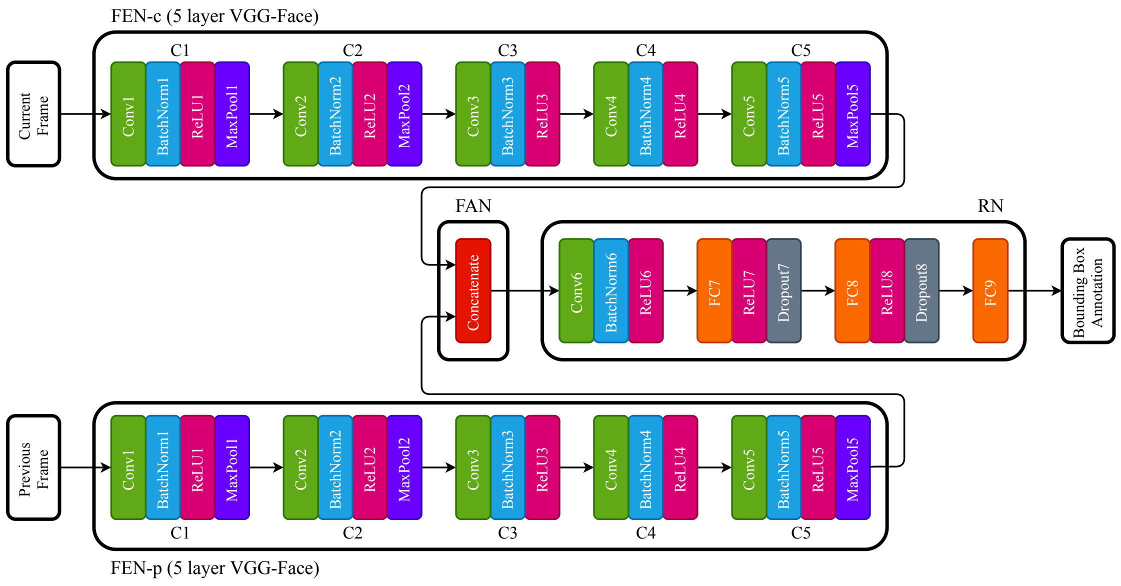


Figure A.2: The network architecture of the $C^{5}_{no\ att}/cp - C^{1} - F^{3}$ tracker.

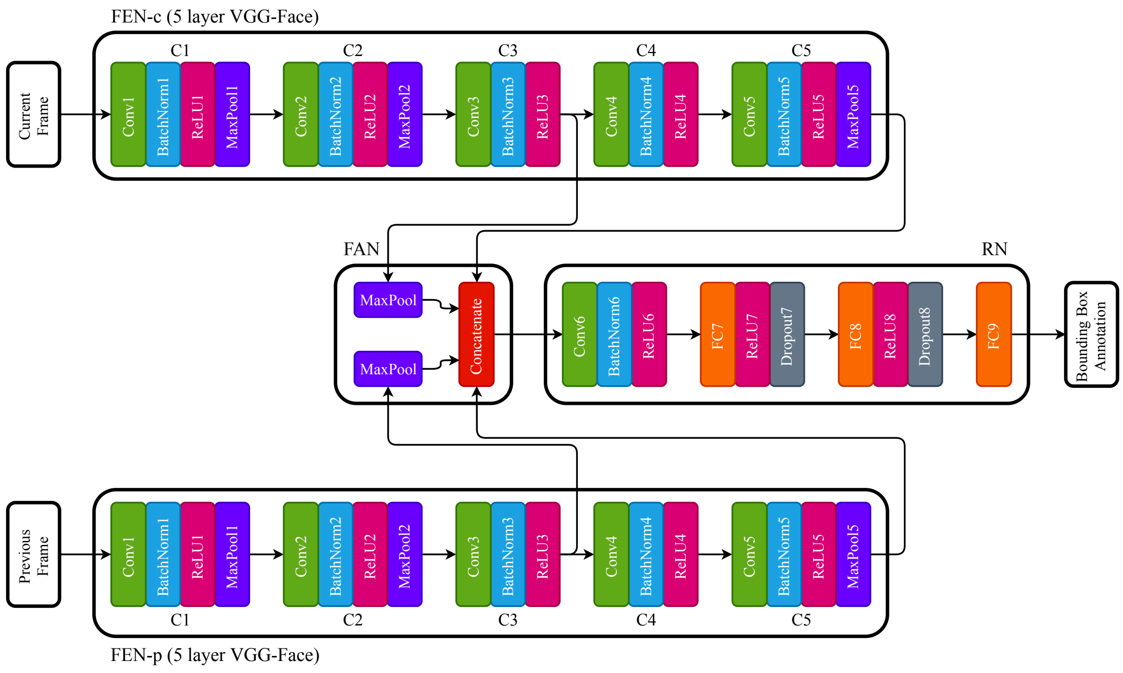


Figure A.3: The network architecture of the $C^{3,5}_{no\ att}/cp - C^{1} - F^{3}$ tracker.

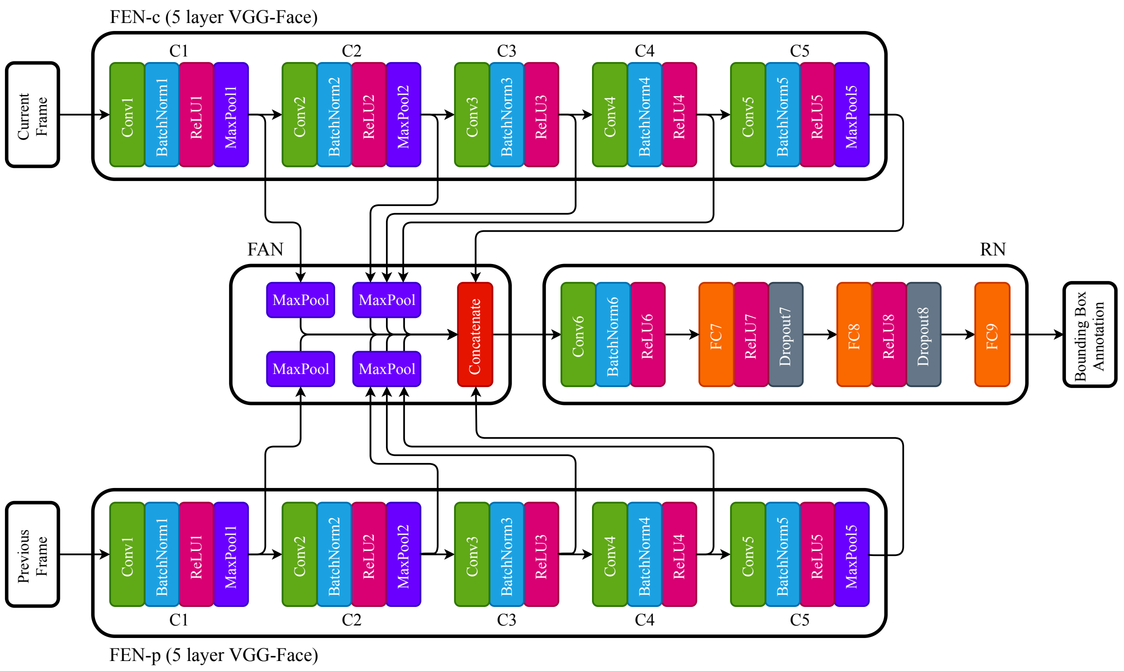


Figure A.4: The network architecture of the $C^{1,2,3,4,5}_{no\ att}/cp - C^1 - F^3$ tracker.

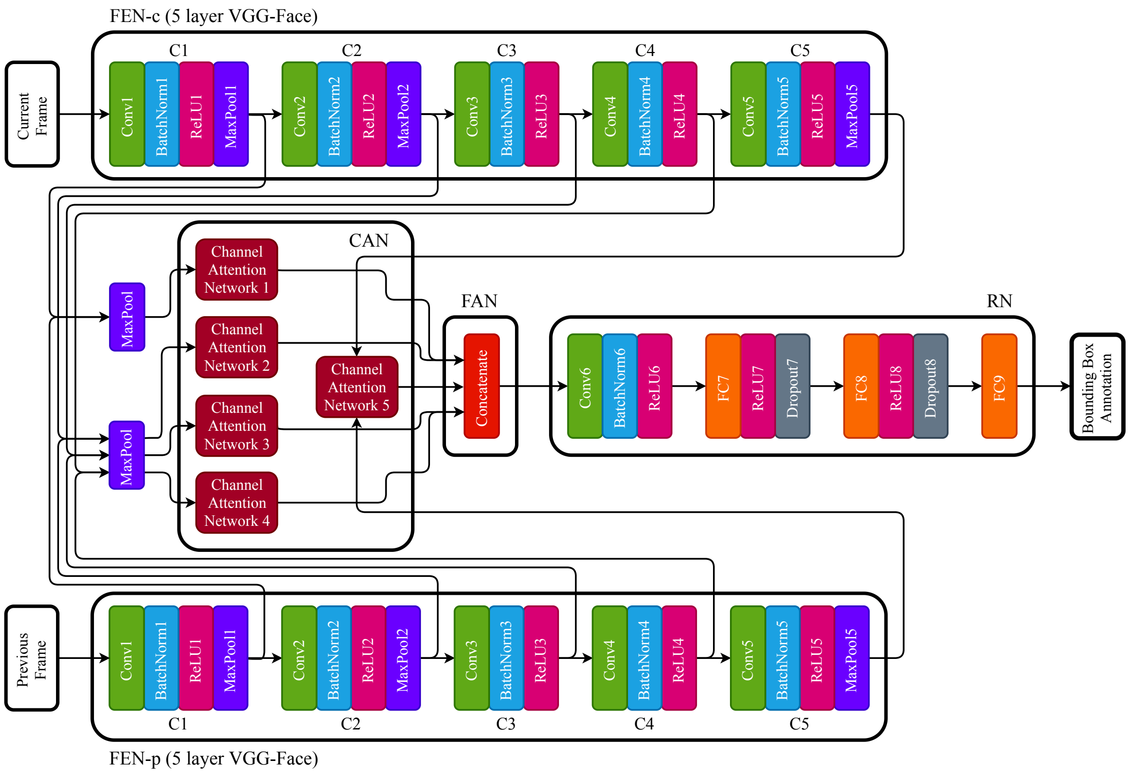


Figure A.5: The network architecture of the $C^{1,2,3,4,5}_{att}/cp - C^1 - F^3$ tracker.