HIERARCHICAL REPRESENTATIONS FOR VISUAL OBJECT TRACKING BY DETECTION

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

BY

BERİL BEŞBINAR

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

SEPTEMBER 2015

Approval of the thesis:

HIERARCHICAL REPRESENTATIONS FOR VISUAL OBJECT TRACKING BY DETECTION

submitted by **BERİL BEŞBINAR** 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. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Eng.**

Prof. Dr. A. Aydın Alatan _____
Supervisor, **Elec. and Electronics Eng. Dept., METU**

Examining Committee Members:

Assoc. Prof. Umut Orguner
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. A. Aydın Alatan
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. Fatih Kamışlı
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. Elif Vural
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. Osman Serdar Gedik
Computer Engineering Dept., Yıldırım Beyazıt University _____

Date: September 10, 2015

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

Name, Last Name: BERİL BEŞBINAR

Signature :

ABSTRACT

HIERARCHICAL REPRESENTATIONS FOR VISUAL OBJECT TRACKING BY DETECTION

Beşbınar, Beril
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. A. Aydın Alatan

September 2015, 115 pages

Deep learning is the discipline of training computational models that are composed of multiple layers and these methods have improved the state of the art in many areas such as visual object detection, scene understanding or speech recognition. Rebirth of these fairly old computational models is usually related to the availability of large datasets, increase in the computational power of current hardware and more recently proposed unsupervised training methods that exploit the internal structure of very large, unlabeled datasets. An exhausting search of good parameters that are usually on the order of thousands, or even millions, is nearly impossible to result in a meaningful model when available dataset is relatively small and this is the reason why deep architectures are barely used for visual object tracking, which is a challenging yet very important task in computer vision. In this thesis, we investigate the use of hierarchical representations within the tracking-by-detection framework, a common strategy in visual object tracking that regards tracking as a detection problem in

still images where temporal information is handled within a Bayesian approach. Stacked autoencoders and convolutional neural networks are trained using auxiliary datasets and the resultant hierarchical representations are experimented both off-the-shelf and after fine-tuning the pre-trained models using the few samples available. Experiments are realized using a challenge toolkit, which not only enables a fair comparison of hierarchical representations with well-known and widely-used hand-crafted features by using the same tracking-by-detection setting, but also demonstrates the performance of utilized framework among all recent visual tracking algorithms. Test results show that exploiting the intricate structure in auxiliary dataset, even without fine-tuning, contributes to the solution of visual object tracking problem.

Keywords: Visual object tracking, tracking by detection, hierarchical representations, deep learning, stacked autoencoders, convolutional neural networks

ÖZ

TESPİT İLE GÖRSEL NESNE TAKİBİ İÇİN SIRADÜZENSEL BETİMLEMELER

Beşbınar, Beril
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. A. Aydın Alatan

Eylül 2015 , 115 sayfa

Derin öğrenme, çoklu tabakalardan oluşan işlemsel modellerin eğitilmesidir ve bu yöntemler görsel nesne tespiti, sahne anlamlandırması veya konuşma tanıma gibi pek çok alanda en son teknolojinin önüne geçmiştir. Nispeten eski olan bu yöntemlerin yeniden doğuşu genellikle büyük veri kümelerinin ulaşılabilirliği, güncel donanımların işlem gücü ve büyük verilerin iç mimarisinden istifade eden, göreceli yeni önerilen güdümsüz eğitim yöntemleri ile ilişkilendirilmektedir. Sayıları binler hatta milyonlar seviyesinde olan iç değişkenlerin iyi olanlarını zahmetli bir şekilde arama işleminin, kullanılan veri kümesinin göreceli küçük olması durumunda anlamlı bir modelle sonuçlanması neredeyse imkansızdır ve bu durum, derin mimarilerin, bilgisayarla görü alanında zorlu ancak oldukça önemli bir hedef olan görsel nesne takibinde nadir olarak kullanılmasının sebebidir. Bu tez kapsamında sıra düzenli betimlemelerin, görsel nesne takibinde oldukça yaygın kullanılan ve takip problemini sabit imgelerde nesne tespiti olarak yorumlayıp

zamansal bilgiyi Bayesçi bir çatı altında anlamlandıran tespit ile takip yöntemi dahilinde kullanımı araştırılmıştır. Yığınlı özkodlayıcılar ve sıradüzensel betimlemeler yardımcı veri kümeleri kullanılarak eğitilmiş ve sonuçta çıkan sıradüzenli betimlemeler hem oldukları gibi hem de az sayıda mevcut olan veri kullanılarak modele yapılan ince ayar sonrasında test edilmiştir. Deneylerin bir yarışma platformu kullanılarak yapılması, sadece sıradüzenli betimlemelerin iyi bilinen ve sıkça kullanılan mühendislik ürünü betimlemeler ile adil olarak kıyaslanmasını sağlamamış, aynı zamanda, kullanılan çatı algoritmanın mevcut tüm takip algoritmaları içerisindeki yerinin görülmesine de olanak tanımıştır. Deney sonuçları, ince ayar yapılmadığı durumlarda dahi yardımcı veri kümelerinin girişik yapılarından faydalanmanın görsel nesne takibi çözümüne katkıda bulunacağını göstermiştir.

Anahtar Kelimeler: Derin öğrenme, evrişimli sinir ağları, geriyayılım, uydu görüntüleri, jeo-uzamsal hedef bulma

To mom, the strongest woman I've ever known...

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Dr. A.Aydın Alatan for his ever-lasting support, understanding and guidance throughout these three years. I feel grateful to him for introducing me such a wide variety of research areas so that I was able to pick the one that matches my interests. I have learned a lot from his eloquence such that he is capable of making ideas and conducted works comprehended with ease and influence, even by those who are unfamiliar with the topic.

Being a part of Multimedia Research Group has always been a pleasure and it is these invaluable friendships that makes the time I spent here remarkable. I would like to thank Emrecan Batı for answering all my questions with patience and understanding. In addition to his technical support and influence in my graduate studies, he introduced me the world of fountain pens, which I become addicted to and taught me how to drive, which made my life a lot easier. However, my deepest gratitude is not for any of these, I appreciate most the opportunity to be provided to trust someone from scratch when I turned in upon myself, and it has been a real privilege to talk as I want when I am together with him, without thinking what others think or say. I am thankful to İlker Buzcu for his company during two years of TA work and all-nighters, as well as for his friendly greetings in the mornings and on the phone. I thank to Yeti Ziya Gürbüz for his support during the projects we were involved in and introducing me such a different perspective about life. Thanks to Ömürcan Kumtepe for being such a kind and naive friend; Akın Çalışkan, if not for anything for his belly laughs, and Ece Selin Böncü, who demonstrates how to be a successful engineer without sacrificing too much from her interests. I also would like to express my thanks to former members of MMRG, Yağız Aksoy, who really knows how to listen and was a great host during my summer in Switzerland, and Emin Zerman, for their friendships and our memorable holiday. Thanks to Ozan Şener, another former

member of our group, for his endless energy and our pleasant conversations. I acknowledge Dr. Osman Serdar Gedik, Dr. Ahmet Saraçoğlu and Dr. Alper Koz, for their precious mentorship. I am also thankful to Enis Kobal, an honorary member of our group, for his valued friendship.

I would like to offer my sincere thanks to Prof. Dr. Pascal Frossard, for his support and trust during my summer internship at EPFL. I had the opportunity to benefit from his experience and wisdom even within such a limited time.

I would like to express my thanks to the fellow members of METU Aviation Society Paragliding Club, which I was enrolled in until the early times of my graduate studies. Spending endless time where there was nothing and nobody else but us, surviving despite scorching hot or perishing cold, it was always a real pleasure to share the moments of amusement when our feet began to tread on air and we only heard the sound of the wind. I also thank to my old friends Caner Bayram and Esra Özban who proved that the friendship is beyond the measures of time and space.

I offer my sincere thanks to Scientific and Technological Research Council of Turkey (TUBITAK), HAVELSAN and ASELSAN for their financial support during my Master's studies.

Finally, nothing would have meaning if I did not have the support of my family. I am grateful to my mother, Zübeyde Beşbınar, not only for providing me such surroundings that I am able graduate from Master's degree today, but also for teaching me how to be an independent woman. Her unconditional love has always given me the strength to stand on my own feet. Although he has not been with me since my first days at school, I am thankful to my father, Mehmet Beşbınar, whose presence and absence I always feel deeply with my heart. I really appreciate being the daughter of such a great man. The last but not the least, I always feel privileged to be a member of such a big, caring family and I express my love to each and every member.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Backpropagation
CAE	Contractive Autoencoder
CNN	Convolutional Neural Network
FNN	Feedforward Neural Network
pdf	Probability density function
PCA	Principle Component Analysis
RBM	Restricted Boltzman Machine
SAE	Sparse Autoencoder
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

Today is the time of search engines that almost *predict* what you have been looking for, or social network websites which *recognize* your friends. We all get used to tools that simultaneously convert speech into text or *intelligent* agents that answer our questions pretty well in real time. These are the outcomes of the passion of mankind to create machines that are capable of thinking, which initiated the academic field of study, *artificial intelligence* (AI). By formal definition, the objective of artificial intelligence research is to design intelligent agents that can perceive the environment and take actions which maximize its chances of success [91]. However, hard-coding the knowledge of environment, which is intuitive and immense for humans, is not straightforward since the exact source of knowledge is not known and not easy to understand. As a solution, research has focused on giving these systems the ability to extract patterns from raw data and acquire their own knowledge, which makes the *representation* of data very important. For decades, artificial intelligence tasks have been attacked by designing right set of specific *features*, which requires careful engineering and domain expertise. With the aim of minimizing human effort and making AI systems adapt themselves for new tasks, methods that allow machines to be fed with raw data and let them discover the required representations have been proposed, and these methods are called as *representation learning*. Similar to the hand-engineered features that do this explicitly, the aim of representation learning is the exploration of the abstractions that lead to the variability in the data, which is known as *factors of variation*.

Deep learning, which has been a very hot topic in the last few years, is also a representation learning method which aims to disentangle the factors of variation by the help of a nested hierarchy of concepts. In deep learning algorithms, multiple levels of representation that correspond to different levels of abstraction are obtained in such a way that complex concepts are built out of simpler concepts by composing simple but non-linear modules one after another. The term *depth* is generally used to refer the number of composition levels, thus the number of updates done to the representation and a more detailed discussion on the term can be found in [11]. The most important aspect of deep learning which provides its wide area of applications is the fact that these compositions are not hand-crafted, rather, they are learned from data using a learning procedure that may adapt itself to any area. In addition, learning from interrelated concepts provide intermediate representations that may be shared among multiple tasks, which fits into the AI tasks that capture different aspects of the same underlying reality.

Learning of hierarchical architectures may be *supervised* or *unsupervised*. In supervised learning, learning algorithm is fed with training data accompanied by target output, which is usually called as label when the target task is classification. On the other hand, in unsupervised setting, algorithms are expected to infer good representation only from the provided data. There are also different approaches for unsupervised learning, and one option is to use data drawn from the same generative distribution of the target task. Such a configuration allows *semi-supervised* learning which refers to exploiting the information from unlabeled data in order to use the resultant representation for the final task by the help of a small set of labeled examples provided after unsupervised learning. It is also possible to use a more general unlabeled dataset which is not restricted to the distribution of the labeled data, which is known as *self-taught learning*. The difference between these approaches may be best explained with an example. Suppose that the objective is to discriminate between the images of cats and dogs. If cat and dog images are provided to an algorithm with corresponding labels, this is supervised learning. In unsupervised setting, the same training data may be provided but without explicitly saying which image

contains cat or dog. Feeding a small subset of cat and dog images with labels after unsupervised learning is known as semi-supervised learning. If a more general dataset, for example, images of different animals, is utilized for learning, this is known as self-taught learning. Although a labeled example provide much more information compared to the unlabeled one, high availability of such general and unlabeled datasets nowadays ensures the interest in both supervised and unsupervised learning algorithms. The amount of unlabeled data may be well depicted by the following facts: by early 2015, about 240,000 images are reported to be uploaded to Facebook every minute, 350 million photos every day and a total number of 250 billion since its foundation [51]. Instagram itself announces that the average number of photos shared every day is 70 million, with a cumulative total of nearly 30 billion. It is 1.83 million public photos per day in average for Flickr in 2014. Such a huge amount of publicly available data brings the opportunity to design machines that are capable of *generalizing* well at tasks that make use of visual data.

Power of deep learning comes from its success in both supervised and unsupervised settings. For supervised learning, deep learning methods usually employ *backpropagation*, a practical application of chain rule that indicates the algorithm how to adjust its internal parameters according to the error at the output. One very well-known and successful supervised learning method is convolutional neural networks, a fairly old architecture that regained its reputation nowadays mostly due to better understanding of the related optimization problem. Unsupervised variations of deep learning, such as stacked autoencoders and deep belief networks, not only serve as a pre-training step for supervised settings but also succeed in obtaining meaningful representations that may be utilized for different tasks. Recurrent neural networks contain feedback loops in their architecture and have an internal memory. Readers are referred to [72] for a brief overview of deep learning and its application areas and more detailed discussions are present in [10], which may be relatively old regarding the pace of the research community.

Although it is nearly impossible to fully exemplify the application of deep learning in a wide range of research areas, some examples will be given here to

demonstrate the power of deep learning not only because results improved the state of the art but also they illustrate suitability to many different areas.

Maybe the most acknowledged application area of deep learning is image recognition. Since 2012, deep learning algorithms dominated the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [100] which is composed of two competitions, namely object detection and image classification. For image classification and localization task, participants are provided a labeled dataset of 1.2 million images that belongs to 1000 categories and asked for the five best decision of their algorithms with corresponding confidence and bounding boxes. Given in Fig. 1.1, the decrease of this top-5 error in classification challenge illustrates the efficacy of deep learning algorithms (convolutional neural networks) with a considerable attack in 2012. The result of a recent work [43] that also utilizes deep learning methods and claims to surpass human performance is also depicted in the figure.

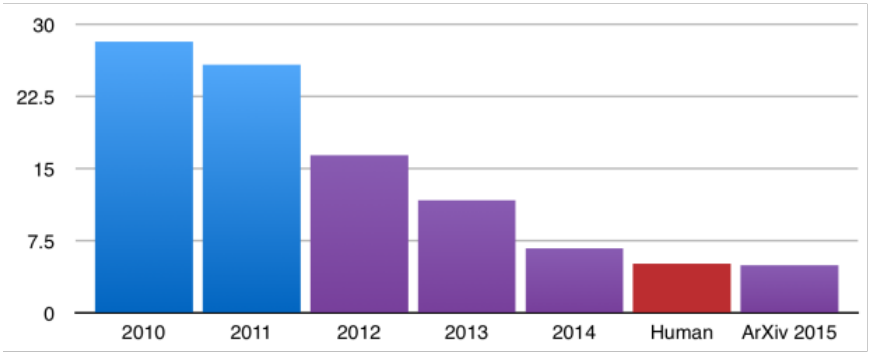


Figure 1.1: Evolution of top-5 classification in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [100]. The success of deep learning algorithms is demonstrated by the decrease in classification error starting from 2012.

Use of deep neural networks also has led to the state of the art results in semantic segmentation problem, which require classification of each pixel in an image. In [129], authors formulated conditional random field as recurrent neural networks and plugged the resultant network into a convolutional neural network for training with backpropagation. The promising results of the proposed method are illustrated in Fig. 1.2.

Another area in which the application of deep neural networks made a considerable contribution is image captioning. An example work in [62] uses a

Figure 1.2: Semantic segmentation result obtained using deep neural networks(image taken from [129])

combination of convolutional and recurrent neural networks to detect and align objects in the image, and generates descriptions of image regions via a multimodal recurrent neural network. Results in Fig. 1.3 demonstrate the success of proposed method.

Figure 1.3: Image caption generation using convolutional and recurrent neural networks (image taken from [62])

As depicted with previous examples, deep neural networks are not only used for discriminative purposes like classification and their generative ability is also utilized in different application areas. For example, in [39], a recurrent neural network synthesizes handwriting and has a demo webpage that enables you to choose a handwriting style to synthesize a sentence you type in, as illustrated in Fig. 1.4

A very interesting work proposed by Gatys et al. intends to create artistic images that combine *content* and *style* of arbitrary images [35] with a pre-trained

Figure 1.4: Handwriting synthesis by recurrent neural networks that conditions its predictions on a typed text sequence [39]

Figure 1.5: Recombination of content information of a photograph and style of an artwork, depicted in bottom left corner of each panel, via convolutional neural networks (image taken from [35])

deep convolutional neural network after minor changes. Authors presented that the reconstruction of input image from different convolutional layers in a network demonstrate different characteristic, in fact, nearly perfect reconstruction is possible from the lower convolutional layers, however, as going to the higher

levels of abstractions, detailed-pixel information is lost whereas the high-level content is still present. Therefore, using feature responses in higher levels of the network and combining them with style representations, they created images whose content information is taken from a photograph and style information from an artwork. Impressive results of the proposed algorithm is depicted in Fig. 1.5.

Unsupervised, generative deep learning methods, such as stacked autoecoders [116] or Restricted Boltzman machines (RBM) [49], are widely used to learn meaningful representations of data, after the discovery of greedy layerwise training trick [12, 49]. The effectiveness of learned representations may be visualized by learning 2-dimensional codes from a network in an unsupervised manner, which actually corresponds to a dimension reduction problem. In [50], authors generated such two dimensional codes of hand-written digit dataset MNIST [74] by well-known Principal Component Analysis (PCA) and a stacked RBM network. Fig. 1.6 illustrates the power of hierarchical representations which cluster data, even in 2-dimensions, without any supervision.

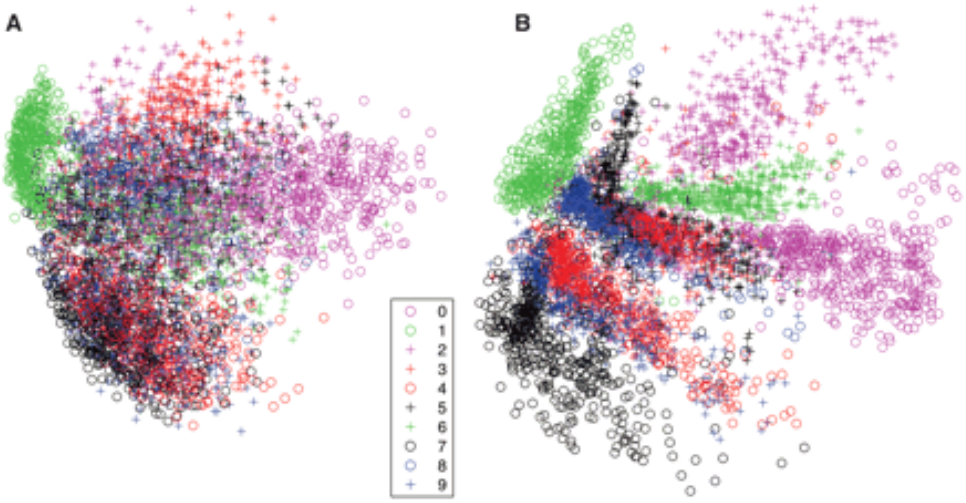


Figure 1.6: Reducing the dimension of data with unsupervised deep learning methods in such a way that they clustered much better in 2-dimensions compared to the reduction realized by PCA.(image taken from [50])

1.1 Problem Statement and Motivation

In this thesis, the ultimate aim is to track a visual object during a video sequence given only the bounding box around the object for the first frame. The problem is usually known as single-object short-term visual object tracking and trackers are expected to contain no information from a pre-learned model of object appearance. However, tracker may adapt itself to a single model for all objects, either pre-learned or not, as long as it does not know what kind of object is being tracked. And similar to machine learning evaluation, if any training is assumed, it should not be on the test set.

If intension is the online learning of an object during tracking, there is only a single labeled data from the initial frame. No other positive or negative examples are present explicitly, and the object is supposed to undergo heavy pose and appearance changes during the motion. If we regard the tracking problem as classification in each frame, it is not only the positive sample (object) that changes significantly, negative samples, i.e., background, may also vary a lot. This specific problem of visual object tracking makes it hard to apply conventional learning techniques to tracking. It is even harder to argue about the application of so-successful deep learning algorithms to tracking problem since these models are composed of hundreds of thousands or millions of internal parameters which are trained on the same order of data, even probably more. However, generalization property of deep architectures that are also assumed to be distributed, i.e., shared across different tasks, may be utilized for per-frame classification. Problem and proposed solutions of visual object tracking are discussed in detail in Chapter 4, and here will be given off some examples to articulate how the representations of deep architectures trained on large datasets are utilized for transfer learning.

An unsupervised and transfer learning challenge is organized in 2011 in order to question capabilities of data representations produced by unsupervised and transfer learning. For the first phase of the challenge, participants are provided with five datasets from various domains, such as Arabic manuscripts or human actions, and they are asked to submit their learned representations (similar-

ity/kernel matrices). Submitted results were then tested for supervised learning tasks by the organizers in order to observe the power of representations learned without any supervision. No labeled data was provided for the first phase. For the second phase, some labels of the same were provided to participants who are expected to improve their representations. However, the test task was similar yet different according to the provided labels in order to evaluate how re-usable the proposed representations were and whether they can be adapted from domain to domain. The winner group of this challenge proposed deep architectures for all of the target datasets [83]. They employed different preprocessing steps for datasets such as standardization, uniformization, contrast normalization or feature selection and utilized deep representations with different single-layer subblocks for different datasets. The indication of proposed work is the fact that stacking different layer-wise representations worked well for all datasets and the best performance is achieved by contractive autoencoder, denoising autoencoder and spike-and-slam Restricted Boltzman Machine. Results also showed that PCA is very effective not only when used as a preprocessing step but also as a last layer. Specifically, the method of transductive PCA which is a PCA trained on the last layer of validation/test set, is claimed to provide retaining dominant variations on the related validation/test set and this step is stated to be decisive for winning the competition.

There are also lots of transfer learning attempts using the representations obtained in the higher layers of deep architectures that are trained on large datasets which relatively differs from the target task. Inspired from the state-of-the-art results presented in recent work [26, 102, 126] that take advantage of transfer learning from higher layers of deep architectures, Yosinski et al. investigated the generality of neurons in the each layer of deep convolutional neural networks [124]. For this purpose, authors randomly divided ImageNet dataset into two, group A and B, with nearly equal number of labels and samples and trained a 8-layer convolutional networks on each dataset, whose structure is the same as AlexNet [67]. Then, they freeze the first n layers of each network where $n \in 1, 2, ..., 7$, initialized the remaining layers randomly and trained both networks on the test dataset, i.e., group B. These networks are denoted as BnB

and AnB, where first letter denotes the dataset on which the initial networks are trained and the second letter stands for the dataset on which architectures are tested. In addition, they constructed two other test networks which are the same as BnB and AnB at the beginning but differs in second training process since all layers are set free for adaptation. These networks with unfrozen initial weights are labeled as BnB^+ and AnB^+. Classification results of all networks on test set of group B, are depicted in Fig. 1.7.

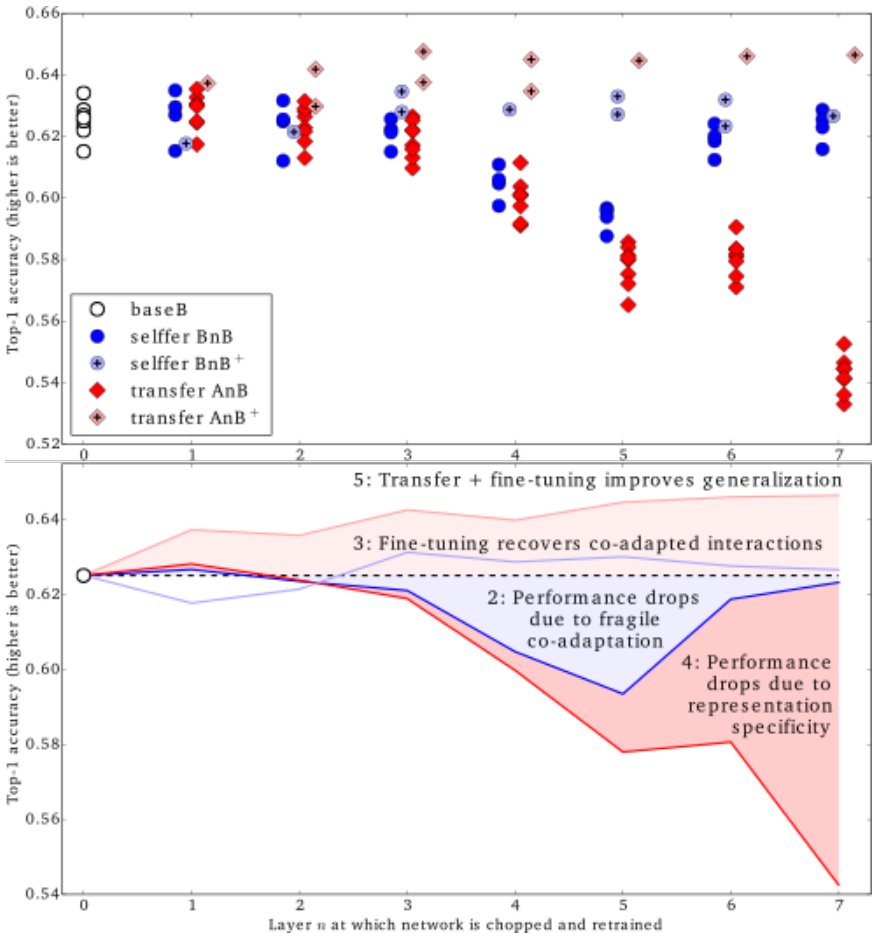


Figure 1.7: Classification results of [124] that indicates how transferable the features extracted at each layer of convolutional neural networks

The strong conclusions of this work [124] can be summarized as follows:

- Features learned in the very first layers are very general and transfering features from the first two layers does not worsen the performance.
- Transfer from middle layers, particularly 4 and 5, causes a considerable performance drop due to co-adaptation of learning features.

- Performance drop due to the transfer from higher layers, 6 and 7, are more related to less general features in the higher levels of abstractions.
- Relearning the output layer that carries the information more related to classification is rather a simple problem compared to feature learning in the lower layers.
- Transfer learning performs better than training from scratch.
- The performance of features that are transferred and then fine-tuned is better than the features trained directly on the target dataset.

As an additional test, they split ImageNet dataset into two groups, one of which contains natural images and the other contains man-made entities. Similar tests conducted on this more variant datasets revealed that features transfer more poorly when datasets are less similar [124].

The work conducted in [92] also proved the representational power of deep convolutional neural networks. The first fully connected layer of *Overfeat* network, which is trained for object classification, ILSVRC13, is tested for a various computer vision tasks. The features are used off-the-shelf, i.e., without fine-tuning, for problems like visual classification, attribute detection, image retrieval, scene recognition or fine-grained recognition. Off-the shelf features presented performance similar to or even higher than state-of-the-art results for all target tasks. Fine-tuning of the network with possible data augmentation beats the state of the art for VOC 2007 classification task and scene classification of MIT-67 indoor scene dataset. Authors claim that CNN features will provide a breakthrough as hand-crafted features did in the last decade.

In [63], CNNs are evaluated empirically on a dataset of 1 million videos that belong to 487 classes. The proposed multi-resolution CNNs are trained over a period of one month and authors compared their video classification results with possible hand-crafted features, which indicated the superior performance of multilayer neural networks compared to linear models. In addition, resultant representations are tested on another large video classification dataset with different configurations: no fine-tuning, fine-tuning of top few layers and fine-

tuning of only classification layer. Best performance is obtained when few top layers are fine-tuned using the new dataset , however, authors propose to fine-tune only the classifier layer when available dataset is small, whether the new dataset is similar to the dataset of pre-trained model or not. In addition, authors remarked that pre-trained networks can be run on images of different sizes due to the parameter sharing.

1.2 Scope and Outline of the Thesis

Deep learning methods have proven their efficacy on a very wide range of domains by learning meaningful hierarchical representations. Both supervised and unsupervised learning capabilities of models that use deep architectures increase the chance of their adoption to new application areas. In addition, tasks that lack large training data to fulfill the requirements of either training or fine-tuning these architectures are shown to benefit from the generality of learned representations. The performance of deep models on domain adaptation problems are considerably good as exemplified on the Motivation part. Therefore, we investigate the utilization of representations obtained by training deep architectures for visual object tracking problem. For this purpose, a tracking by detection framework, which consists of representation, classification and motion model modules, is constructed. Tracking problem is handled by classifying the representation of candidate image patches, which are proposed by a motion model, and this modular structure allows experimentation with different representation algorithms. Classifier in the framework is trained using the single labeled data provided in the initial frame. Once a new frame arrives, candidate image patches are processed by representation module and resultant *features* are fed to the classifier, which decides how likely each image patch corresponds to the new bounding box of the target object. The motion model, which is a recursive Bayesian filter with *predict* and *update* stages, utilizes the corresponding probability values for each image patch for its update step.

We experimented with two different deep architectures, namely convolutional neural networks and autoencoders, as representation modules and they are ex-

plained in detail in Chapter 3. Due to the lack of training data in visual object tracking, features from a pre-trained CNN are applied off-the-shelf after an addition of classification layer on the top. For autoencoders, we exploit auxiliary data that contain different objects and fine-tune the whole model using the few samples available for tracking. Performance of learned representations and some well-known hand-crafted features within the proposed tracking by detection framework are compared for the completeness of the experiments.

Chapter 2 presents a brief survey of visual object tracking by summarizing different aspects and demonstrating current approaches. Results of recent benchmarks and challenges on visual object tracking are given in the same chapter in order to demonstrate the performance and variety of approaches for the solution of the problem. Brief explanations of some best-performing algorithms are also provided. Different datasets and performance measures are available for visual object tracking and the ambiguity of evaluation is depicted by the difference between the performance of the same algorithm on different evaluation methodologies. Dynamic performance and ranking results throughout recent years not only illustrate the difficulty of proposing a generic and robust solution, but also demonstrate the interest in the problem.

In Chapter 3, a thorough background information of deep learning algorithms related to convolutional neural networks and autoencoders is provided.

Chapter 4 first introduces the few approaches to attack visual object tracking problem with deep learning. The experimentation scheme and algorithms used for performance comparison are also given in Chapter 4 and it summarizes the results of experiments realized using a challenge toolkit.

Chapter 5 concludes the presented work by related inference and possible future directions.

CHAPTER 2

VISUAL OBJECT TRACKING

Visual object tracking can be defined as the automatic estimation of the trajectory of a moving object over a sequence of images. Object tracking is an important task with a wide range of application areas such as video surveillance for security, human-robot or human-computer interaction, traffic monitoring, driver assistance, vehicle navigation, video compression, sports video analysis, augmented reality, etc. Although humans can easily detect and track objects, automatic tracking of objects in real world data can be very complex due to cluttered background, fast and abrupt motion, large target appearance variations caused by illumination, pose and/or viewpoint changes and partial or full occlusions.

There is a substantial work for tracking some specific objects such as face [56] or human [3, 36]; however, in this thesis, the term "object" will be used for a rather general purpose to indicate standalone things with well-defined boundary and corresponding center. The key components of object tracking can then be described as object initialization, representation of the object and motion estimation. Initialization of the tracker may be manual and object location can be initialized by a user in the first frame using a bounding box. Otherwise, initialization is realized by automatic object detection algorithms. Representation of the object refers to the mathematical modelling of object descriptors which is deeply investigated in [76] and will be summarized in the following sections. Motion estimation is the step in which the trajectory of the target is predicted using the current, and possibly previous frames.

2.1 Object Representation

Choosing a proper representation for the target not only plays an important role in the success of the tracker but also directly affects the motion estimation step. Objects can be represented by their shapes, appearance or both. As categorized in [123], shape representation can be achieved by points, geometric or articulated shapes, skeletal models, silhouettes or contours, which are illustrated in Fig. 2.1 On the other hand, appearance models may be estimated either parametrically, for example, using Gaussian or a mixture of Gaussians, or non-parametrically such as histograms. If not estimated, the appearance models can be templates formed using simple geometric shapes or silhouettes. Another option is to learn the appearance of the object using training data and update the model simultaneously during tracking by storing the object features such as texture, color or gradients. Note that the appearance model can be single view or multi-view.

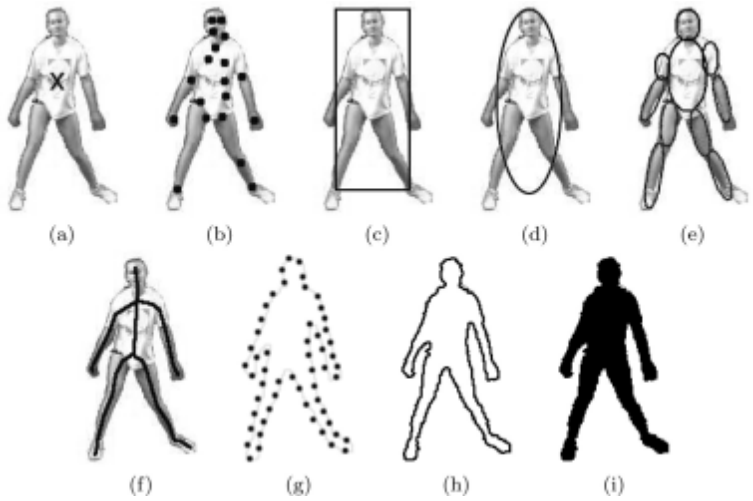


Figure 2.1: Illustration of object representations: (a) centroid, (b) multiple points, (c) rectangular patch, (d) elliptical patch, (e) part-based multiple patches, (f) object skeleton, (g) control points on object contour, (h) complete object contour, (i) object silhouette (image taken from [123]).

In addition to the way the object is represented, selection of features to be used during tracking is also very important. Color, texture, optical flow or edges are the features that are generally used in object detection and tracking algorithms [123]. Color is a physical property of the object and is influenced

by its surface reflectance properties as well as the illumination. Texture is less sensitive to illumination changes and measures the variation of intensity between neighbouring pixels and is an indicator of object surface properties. Optical flow defines the translation of pixels in a region between consecutive frames and tries to capture the spatio-temporal motion information of an object. Edges, which refer to strong changes in image intensities, are usually used to describe object boundaries and also less sensitive to illumination changes.

Raw pixel intensities, boundaries, texture, optical flow or even histograms of color and/or texture can be used to describe the objects globally. As another option, local descriptions of objects are also possible using local templates, segmentation, saliency or point detectors. Local templates are similar to the global ones except being more robust against occlusions and shape changes. Being used for either object boundary or superpixels, segmentation aims to partition an image into visually similar regions and may be realized by algorithms such as mean-shift clustering [21], graph-cut [16] or active contours [19] according to the need for description. In the case of superpixel generation, the aim may be to build a dictionary of superpixels which corresponds to local templates. On the other hand, point detectors are used with an aim to find interest points that has an expressive power of description with respect to their local neighbours. Harris interest point detector [41], Scale Invariant Feature Transform (SIFT) [79], Speed Up Robust Features (SURF) [8], Histogram of Oriented Gradients (HoG) [23] and their variants are frequently used for object detection and readers interested in performance evaluation of local descriptors are referred to the related survey [84]. Finally, saliency detection is mostly used to exploit the salient information in the image which is believed to be distinct and robust so useful for object detection.

Both local and global features have their own advantages and disadvantages. Although being susceptible to appearance changes, global features are simple, computationally efficient an therefore, easy to track. Local features are more robust to deformations, partial occlusions and illumination changes since they capture local structures; however, they miss the global structure and are afflicted by noise and background distraction. Therefore, it is more common to track

objects after a detection step instead of tracking the features directly.

2.2 Object Detection

Apart from the tracking aspect, object detection is a standalone problem in computer vision with other application areas such as recognition and scene understanding. Object detection approaches tend to use a statistical model which makes use of the image features that are briefly explained beforehand. According to the model generation process, these approaches can be classified into two groups as discriminative and generative. [76] The purpose of generative approach is to reveal a model which represents the data best so that the way the object is re-generated results in the possible smallest error. On the other hand, the objective of the discriminative models is to separate the object from non-object regions, therefore, they regard the detection problem as a binary classification problem. Therefore, discriminative models try to capture the structural information not only in the object but also in the background. Both discriminative and generative models use online learning techniques during tracking so that the models are adapted to appearance changes and occlusions.

One generative model approach is to use mixture models, such as Gaussian, in order to represent the object using raw pixel intensity values or local features. Mixture models require the number of components in the mixture as an input to the algorithm; therefore, heuristic determination of the number of components may lead to deterioration of the tracking performance. Unlike mixture models, kernel-based methods use kernel-density estimation for modeling the data which may be based on color, shape or spatio-temporal distribution information. However, computational complexity and memory consumption are important drawbacks for kernel-based tracking algorithms as well as the difficulty of determination or learning of the kernel. In subspace learning-based generative appearance models, target object is associated with several underlying subspaces and is expressed as a linear combination of basis functions of these subspaces. Linear subspace models either construct vector-based models for lower order or models based on a matrix or tensor for higher orders, whereas

non-linear subspace models assume that the training data lie on a non-linear manifold and use non-linear dimensionality reduction techniques such as Local Linear Embedding (LLE) and kernel principal component analysis (KPCA). Sparsity constraints and spatial pooling schemes are also introduced for higher efficiency. Addition of subspaces, i.e., multi-subspace learning is also possible for the increase in expressive power with an additional cost of memory and computation. Regarding the temporal coherence of frames, autoregressive models are also used for subspace learning methods.

In contrast, discriminative models try to separate the object from background using classification methods. For example, boosting methods use simple (weak) classifiers that use features with weak classification performance in order to increase the accuracy by a single, strong classifier that is composed of simple ones. Self-learning boosting-based models first train a classifier using a set of positive and negative samples obtained from the classification result of previous frames and use that trained classifier for object detection in the current frame [38]. Semi-supervised learning is applied to boosting-based methods in order to avoid the model drift due to the error accumulation in self-learning boosting-based models. Another classifier used for discriminative models is Support Vector Machines (SVM), which aims to find a hyperplane that separates the two-class data best using the most informative samples called "support vectors". Selecting an effective kernel and computing it efficiently are the two key factors that affect the performance of SVM classifiers. Like boosting-based methods, SVMs can use self-learning or co-learning techniques. The main disadvantage of self-training SVM-based discriminative model is that it does not use the discriminative information from unlabeled data or multiple sources. On the other hand, co-learning SVM-based models require several labeled samples for semi-supervised learning methods, which is not applicable for many realistic scenarios. A more efficient model for discriminative methods is the appearance models that are based on randomized learning techniques, which aim to construct a classifier ensemble by selecting input and/or features randomly [76]. Although these methods are eligible for multi-core or GPU implementation so that the overall run-time is reduced, their performance may be unstable depending on the tracking scene.

One other alternative is to use discriminant analysis methods which seek a lower dimensional feature space where the separability of classes is increased. Popular linear discriminant analysis assumes Gaussian distribution of classes and the kernel trick can be used to learn non-linear mappings [76]. Graph based learning methods, which need large number of labeled data, are also used for discriminant analysis.

Discriminative and generative models have their own weaknesses and strengths and hybrid models are used to take the advantage of the strengths of both models by a complementary use. This combination may be realized in a low-level feature extraction step, in high-level decision step or in both steps [76]. Another way to combine discriminative and generative models is to use the output of one model as the input of other.

2.3 Motion Estimation

Object detection and establishing correspondence between object instances in consecutive frames can be performed jointly or separately. In general, the joint solution offers iterative update of the object region and location. Otherwise, the object detection is performed for each frame and the tracking algorithm generates the trajectory by corresponding the detection results.

2.3.1 Motion Estimation for Point Representations

The tracking approach heavily depends on the choice of object representation. If point representation is chosen for the detected object in each frame, points can be associated with methods that use motion heuristics such as proximity, maximum velocity, small velocity change, common motion, rigidity or proximal uniformity [123]. A cost for the association between points on consecutive frames is defined using these motion constraints and the correspondence is established by minimizing the defined cost. However, statistical models proposed for point tracking aim to model uncertainties and utilize state space approaches for object modelling. There is a variety of options for the object state; however, position,

velocity and acceleration are the object properties that are mostly used for tracking algorithms. The objective is to estimate the current state of the object using all measurements up to the current moment, which is equivalently estimating a conditional probability density function. Bayesian recursive filters with *prediction* and *correction* steps are widely used to solve the problem [123]. In case of multiple points to be tracked, a step for the association of measurements with the tracks is required.

Using a Kalman filter for the estimation of the object state is a very common approach. Kalman filter assumes the initial state is known and is distributed by a Gaussian as well as a Gaussian noise. In the prediction step, a linear state model is used to predict the current state of the object. The predicted position is updated in the correction step that uses the current position obtained using the object detection algorithm. There are extensions of Kalman filter for non-linear motion models: extended Kalman filter uses Jacobian for the linearization of the underlying non-linear model, whereas unscented Kalman filter makes use of a deterministic sampling method whose output sample points are utilized for the mean and covariance estimation. Kalman filter is optimal for the valid assumptions on distributions and noise and extended and unscented Kalman filters are useful when the non-linearities are mild and uncertainties are small [123]. If the assumption of normally distributed state variables and noise is not possible, particle filtering can also be used for state estimation. Particle filter is a sequential Monte Carlo method that represents the posterior probability density function by a set of samples called "particles" with associated weights and computes the estimates using these samples and weights.

As stated earlier, multiple target tracking requires a step where the detection data is associated with each track. One option is to solve correspondence problem using deterministic methods such as nearest neighbour algorithm; however, deterministic methods usually fail in case of close objects or occlusion. On the other hand, there are several statistical approaches to overcome data association problem. One well-known technique is Joint Probabilistic Data Association Filter (JPDAF), which associates measurements to tracks and the most probable update is achieved as a combination of all potential assignments. However,

JPDAF is not able to handle objects entering or exiting the field of view since the number of tracks is assumed to be constant. Another proposition for correspondence problem is Multiple Hypothesis Tracking (MHT) [93], which propagates alternative data association hypotheses in case of assignment conflicts and decision is taken after several frames. However, MHT algorithm is computationally exponential in memory and time.

2.3.2 Motion Estimation for Appearance-Based Models

If the object is represented using appearance models instead of a point, the motion of the object, usually in parametric form, is computed for tracking. If a template-based appearance model is used for object representation, template matching, which seeks for a region in the whole image that is similar to the object template according to a similarity metric, is a widely used method for the tracking purpose. There are several variants of template-matching, some of them narrows down the search area with proximal uniformity assumption and some variants propose the use of different features such as image gradients, color histograms or mixture models instead of the using intensity directly. One well-known appearance-based tracking algorithm is Kanada-Lucas-Tomasi (KLT) tracker [104] whose goal is to compute the translation of a rectangular region centered on an interest point. The translation is computed iteratively and the decision to continue tracking features is made according to the quality of those features that is computed as sum of square differences after the affine transformation between regions in consecutive frames. Another option to track objects represented by appearance-based models is to learn the multiple views of the object offline and use them for tracking. Instead of describing objects with simple geometric shapes, some approaches prefer using silhouette-based models in the form of a density functions (edge or color histograms), object edges, object contour or any combination of them. Silhouette is usually detected by background subtraction and silhouette-based modelling leads to trackers with shape matching or contour tracking. Shape matching is very similar to template matching: it searches for the chosen model of the silhouette in the current frame, but the model is updated every frame to handle appearance changes. On the

other hand, contour tracking methods evolve the initial contour of a detected object using state space models, which is defined in terms of the motion parameters and the shape, or minimizing a contour energy that involves temporal information. Silhouette-based tracking methods can handle a variety of object shapes as well as objects that split or merge; however, occlusion handling is only achieved by motion models that assume constant velocity or acceleration.

2.4 Online Object Tracking: A Benchmark [121]

In 2013, Wu et al. published a benchmark [121] and evaluated 29 online object tracking algorithms on a dataset which consists of 50 sequences. Initial position and size of the object is provided to the trackers and robustness of the algorithms is tested by perturbing the initial bounding boxes both temporally and spatially.

Wu et al. first review the algorithms regarding their object representation, search mechanism and model update approaches. Similar to brief explanation of object representation approaches provided beforehand, authors refer to the works that use raw intensity values, subspace models, sparse representations, visual features, discriminative models, learning models and multiple representation schemes. They also categorize the search mechanism as deterministic or stochastic. Results of the benchmark emphasize the importance of updating the object representation and/or model so that appearance changes are not problematic. Benefit of involving the context information in the tracking algorithm is also emphasized for occlusion handling.

For performance evaluation and analysis of algorithms, sequences are annotated with attributes of illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-view, background clutters and low-resolution so that any algorithm can be evaluated on a sequence with a specific attribute for the evaluation of the performance of the tracker on that specific problem.

One quantitative metric authors use for the analysis is precision plot, which indicates the percentage of frames in which the estimated location is close to

the ground truth within a given threshold. Precision plot is obtained by sweeping that threshold and re-evaluating a specific tracker on sequences. It is claimed to cope with the ambiguity of well-known evaluation metric of center location error during the frames when the tracker loses the target.

Another evaluation metric is the success plot which shows the percentage of frames when the bounding box overlap between the algorithm and the ground truth is larger than a given threshold. Algorithms are ranked by the areas under success plot curves. In order to measure the robustness of the algorithms to initialization, authors conducted tests during which the tracking algorithm is started at different frames with different bounding boxes, which accounts for the error of object detection algorithm in case of automatic detection.

Figure 2.2 shows the performance of top ten algorithms in terms of precision and success plots when the initialization is made with ground truth scale and position, and average precision or success is taken into account, which is called one-pass evaluation (OPE). The ranking is according to the area-under-curve for success plots and an error threshold of 20 for precision plots.

Figure 2.3 illustrates spatial robustness evaluation (SRE) of top ten algorithms when trackers are evaluated for 4 center shifts, 4 corner shifts and 4 scale variations of the initial bounding box.

Figure 2.4 presents the temporal robustness evaluation (TRE) of top ten algorithms for which sequences are partitioned into 20 segments and trackers are tested for all segments individually.

Notice that one-pass evaluation is one instance of both spatial and temporal robustness evaluation.

The best algorithm in both temporal and spatial robustness evaluation, but the second best in one-pass evaluation is Struck [40]. Hare et al. propose an adaptive tracking by detection scheme which links the learning and tracking problems instead of labelling positive and negative samples according to the object detection result and updating the classifier separately. In other words, in typical tracking by detection algorithms, tracker estimates the current position

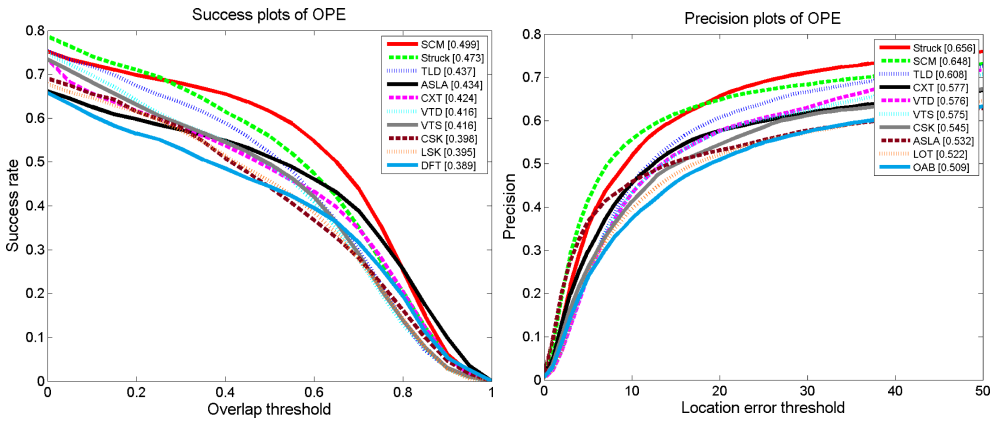


Figure 2.2: Success and precision plots of top ten algorithms for one-pass evaluation [121]

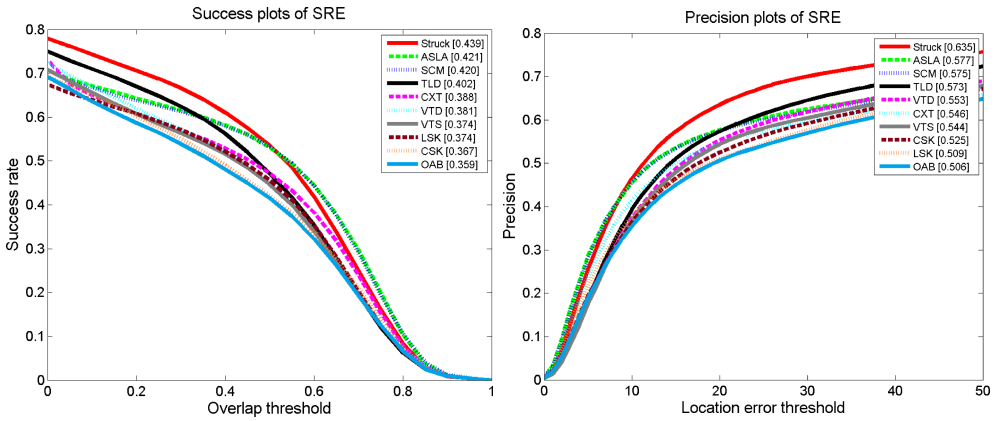


Figure 2.3: Success and precision plots of top ten algorithms for spatial robustness evaluation [121]

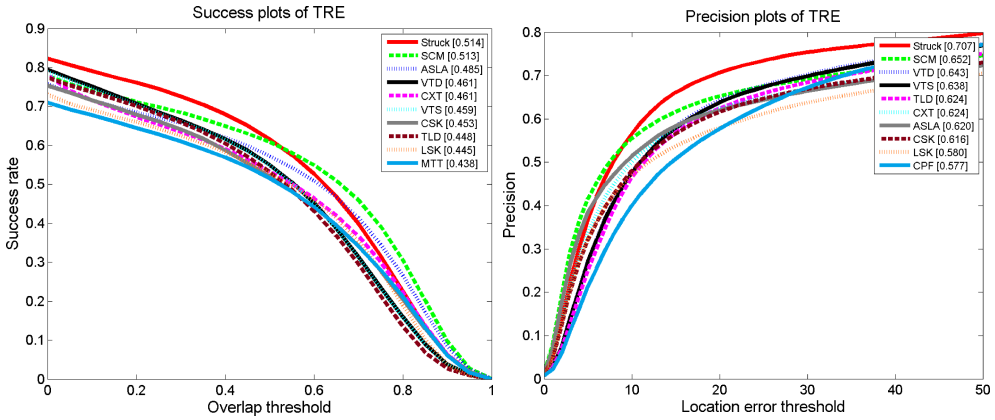


Figure 2.4: Success and precision plots of top ten algorithms for temporal robustness evaluation [121]

of the object by making use of a confidence function, generates a set of training examples with binary labels by first sampling with different transformations and

then labelling with the classifier. The classifier is updated using the samples with binary labels, without giving priority to any of the generated samples. In the proposed framework, instead of binary labels, a prediction function that aims to estimate the two dimensional translation between frames is learned with structured-output SVM. Therefore, labelled examples that consist of both image patch and transformation relative to the current location are provided to update the prediction function. In order to avoid the increase in computational cost by the provided support vectors each step, a budget algorithm is proposed which is based on the removal of the support vector that leads to the smallest change to weight vectors in structured-output SVM. The performance of the algorithm is shown to deteriorate when the overlap threshold for success plots is increased or location threshold for precision plots is decreased and it is mainly due to the fact that the algorithm predicts the location change and is not designed to handle scale changes.

The top ranked algorithm in OPE is a sparsity-based collaborative model (SCM). SCM [130] uses both holistic templates and local representations and detects objects by the help of a sparsity-based hybrid model. Discriminative classifier is based on the training samples of down-sampled gray level image patches. Image patches are converted to vectors and positive samples are selected around the manually selected object whereas the negative samples are also close to the object and may even contain small parts of it. Then, determinative features are selected from these samples for both foreground and background via a sparsity constraint and original feature space is projected to the space spanned by these selected distinctive features. On the other hand, gray level vectors of overlapped windows is used for the generative model and a dictionary is generated using the patches within the initial bounding box. Another sparse vector is obtained for each patch regarding the generated dictionary and histogram of the sparse vectors is computed. One important point is that the patches with large reconstruction error is regarded as occlusion and discarded for histogram computation. The collaborative model is then constructed for the candidates taken around the detection result of the previous frame with a particle filter. Likelihood function for each candidate is computed using both the reconstruc-

tion error for foreground and background models in discriminative classifier and the similarity of histograms between the candidate and the template for generative model. In order to cope with appearance changes, negative templates in discriminative classifier and template histogram in generative model is updated. There is no change on the dictionary during tracking process in order not to get affected from tracking errors. Authors relates the performance of the algorithm to the unification of discriminative holistic and generative local features.

Despite of the relatively poor performance on temporal robustness evaluation tests, TLD [60] achieves good results in one-pass evaluation tests. Kalal et al. propose a semi-supervised learning process guided by positive and negative constraints, called P-N learning, in order to learn a object detector during tracking. They use image patches obtained by scanning a window throughout the entire image, regard these patches as unlabeled data and claim that patches are related both spatially and temporally so they have spatial and temporal structure. Based on the fact that an object appears in one location only, the trajectory is claimed to define a curve in the video volume. Randomized forest classifier is used as a detector and trained using the patches within the initial bounding box. Ferns of the classifier computes 2-bit binary patterns of a number of images patches which results in a feature vector on the leaf nodes. These feature vectors correspond to posterior probabilities, which are averaged later and average value higher than 50% results in a positive detection result. For each frame, classifier and Lucase-Kanade tracker find the location of object. Confidence between the initial frame and last tracked frame is calculated and if the trajectory is validated by the confidence value, P-constraints that require patches close to trajectory to have positive labels and N-constraints that forces patches far away from the trajectory to have negative labels are applied. Positive and negative examples are used to update the trajectory. However, re-initialization is required in case of a strong detection far away from the trajectory. Note that the ranking of this algorithm is lower in TRE since it performs better on longer sequences as a virtue of this learning scheme.

Adaptive structural local sparse appearance model (ASLA) [58] also achieved good results in all of the evaluation tests. In order to obtain a local sparse

model, a dictionary is generated using the overlapped local image patches inside template target regions. A sparse coefficient vector is obtained for local patches inside every target candidate and these coefficient vectors are divided into segments. Segment coefficients are then averaged to obtain weights which gives more importance on more frequent ones. In order to capture the structural characteristics using modified coefficient vectors, an alignment pooling scheme is proposed. Using the similarity between pooled features in consecutive frames as observation model, tracking problem is attacked within a Bayesian inference framework. In addition, a template update scheme, which utilizes not only the last tracking result but also earlier ones, is proposed with a random selection of template to be updated. Like many other tracking approaches, model update procedure is inspired from the incremental learning method proposed in [97], which is also evaluated in benchmark and not ranked in top ten in the given results.

As can be concluded from the given few examples, there is a variety of approaches for visual object tracking. The large scale experiments carried out and summarized in [121] indicate that sparse representations are powerful for appearance changes and local sparse representations have shown better performance than holistic approaches. In addition to structured learning approaches, sparse representations are also effective for handling occlusions. Readers are referred to [128] for a survey that reviews the tracking algorithms that are based on sparse coding. Authors of the benchmark also conclude that the performance of stochastic methods is insufficient in case of fast and abrupt motion mainly due to poor dynamic models, which is advised to be improved.

The work of Wang et al. [117] is not included in the benchmark since it is published the same year as the benchmark. Authors propose a non-negative dictionary learning algorithm which can automatically detect occlusions. Proposed method is similar to L1 Tracker [82] that represents the object as a sparse linear combination of object and trivial templates and then uses these coefficients within the particle filter framework. Wang et al. first find a robust sparse coding using dictionary templates by introducing a Huber loss function. Later, the template update is also formulated as a dictionary learning problem where

a matrix composed of previous tracking results is used and approximated by a low-rank component. Authors claim that the engagement of previous tracking result and using a sparse representation forces the tracker to discard the detections with occlusions in model update. In order to further increase the robustness, background templates sampled randomly from previous frames are augmented to the dictionary. Instead of holistic templates, authors prefer using L1 regularized logistic regression for feature selection. Test results show that the algorithm presents a better performance than some approaches included in the benchmark.

2.5 Visual Object Tracking (VOT) Challenge

In addition to the benchmark published in 2013, a Visual Object Tracking (VOT) workshop has been organized yearly since 2013. The aim of the workshop is to provide a common platform to visual tracking community where an annotated dataset is available with an evaluation methodology.

The workshop in 2013 was in conjunction with International Conference on Computer Vision (ICCV) 2013, for which a total of 27 trackers, which includes eight baseline trackers, are evaluated on the benchmark dataset. The competitors in the challenge were casual, single object visual trackers that only require the first frame to be manually initialized. During evaluations, trackers are re-initialized manually in case of a failure, which means zero overlap between the detected and ground truth bounding boxes. The dataset of 16 sequences are selected from a pool of sequences that had been used for visual object tracking evaluation [64]. Each frame of the sequences in the dataset is labeled with five visual attributes that are occlusion, illumination change, motion change, size change and camera motion. Two orthogonal measures are said to be chosen for the performance: accuracy and robustness. Accuracy is computed to find the overlap between the bounding box predicted by the tracker and the ground truth whereas the number of failures is counted in order to find the robustness of a tracker. After failure of a tracker in a sequence, user manually reinitializes the tracker 5 frames after the failure to continue evaluation of the tracker for the remaining

part of the sequence. Three experiments are conducted for each tracker: for the first experiment, the first frame is initialized by the ground truth bounding box. Secondly, trackers are tested using bounding boxes perturbed randomly in position and size. The third experiment is the same as the first one but frames are converted to gray-scale. Trackers are tested on each sequence by being run 15 times. Accuracy and robustness of a tracker are first computed as the average accuracy or robustness over a frame in a sequence, then, the average over frames is calculated in order to evaluate the performance of that specific tracker on a single sequence. Trackers are ranked separately on each attribute sequence regarding their accuracy and robustness, and the ranks are averaged with equal weight over different attributes. The final ranking on each experiment is obtained by taking the average of two rankings. During the ranking of trackers on attribute sequences, groups of equivalent trackers are determined so that the rank of a tracker is corrected by an average of the ranks in the group of equivalent trackers. Readers are referred to the resulting report [64] of the challenge for the explanation of how the equivalence of trackers is determined.

Performance of algorithms participated in 2013 challenge is summarized in Figure 2.5, where each tracker is represented by a point in joint accuracy-robustness rank space.

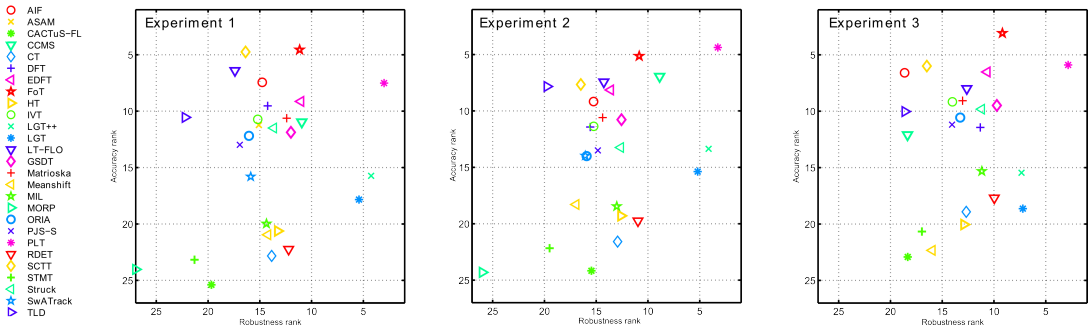


Figure 2.5: Ranking results of VOT 2013 challenge [64]

The best performing method in 2013 challenge is PLT, which is introduced as a single scale, pixel based lookup table tracker in the challenge report. Heng et al. proposes using the sparse structural SVM in Struck algorithm [40] to select discriminative binary features that are constructed from gray scale, color and gradient information. During training, features are weighted according to a

probabilistic object-background mask obtained from color histograms. Since the algorithms works in a single scale, it has difficulties in adaptation to the target size.

For the second best algorithm on average, FoT, Flock of Trackers, Wendel et al. utilizes a number of independent trackers that are initialized on a regular grid inside the specified object frame. These independent trackers are able to change position within cell regions. Object motion is estimated as a combination of local trackers that are expected to represent translations. The estimation is done after filtering out the trackers that do not satisfy neighbourhood consistency and Markov model constraints. Although experiments show that FoT is the best performing algorithm regarding the accuracy, lowered robustness compared to PLT results in a second best rank on average.

Another good performing algorithm in the challenge is EDFT, an enhanced field tracking algorithm that uses channel representations for an efficient computational scheme [31]. EDFT is a variation of Distribution Field Tracking approach [103] that uses smoothed local histograms for region-based tracking with a density-based comparison. Use of channel representations instead of smoothed histograms is claimed to be more efficient yet powerful with the appropriate parameters derived in the paper.

LGT++, an enhanced adaptive coupled layer tracker [122] is an adaptation of LGT [17], which employs both local image patches and a probabilistic global appearance model within an interactive framework. In [17], image patches are tracked using Kalman filters and validated using matching strategies. Patches that could not be matched are removed from the model whereas new ones are allocated using the global appearance model that uses colour, motion and shape information of the candidate target. In LGT++, the previous algorithm is improved with adaptive patch size, a drift avoidance method based on marginal density, a memory recovery for occlusion handling and particle filter to collaborate with Kalman filter. The results of the algorithm on VOT dataset shows that it is quite robust under noisy initializations and the accuracy of the tracker is reasonably well.

The top algorithm on the benchmark [121], Struck placed after ten algorithms on average despite of its success on occlusion handling test. Another algorithm that achieved good results on the benchmark, TLD [60], got a poor place on the challenge ranking. These results emphasize the importance of evaluation metrics and methodology, as well as the need for a common platform to evaluate and compare visual tracking algorithms.

The workshop organized in 2014 also intended casual, model-free trackers that use sequences from a single camera to track single object and total of 39 trackers are evaluated for the challenge. The new dataset is obtained from a pool of 193 sequences and final 25 sequences are in color and longer than 200 frames [65]. Per-frame annotation with five visual attributes was the same as in 2013; however, rotated bounding boxes are used for elongated, rotating and deforming targets in some sequences. The first two experiments in 2013 challenge were conducted the same way for 2014 challenge and ranking of algorithms was realized using the accuracy and robustness criteria as in 2013. However, equivalence of trackers is determined by comparing the average accuracy per frame of trackers within a threshold, computation of which can be found in [18]. In addition, a normalized time-measure is introduced for a more fair comparison of execution time of trackers.

The evaluation results of VOT 2014 challenge are summarized in Figure 2.6 on the same accuracy-robustness rank space.

PLT 13, which was represented as PLT for VOT 2013 challenge, and its successor, PLT 14 with a difference of size adaptation, are the best two algorithms in terms of robustness. However, regarding the accuracy, best performing algorithms are DSST [24], SAMF [77] and KCF [47], the last two of which are variations of CSK tracker, a circular structure kernel tracker [46] that also achieved good results on the online benchmark [121] with the highest speed. CSK tracker makes use of the structure of training samples obtained by cyclically shifting stacked image patches and link the resultant circular matrix to Fourier analysis for a kernel learning problem as efficient as linear classifiers [46].

DSST is discriminative scale space tracker [24] that uses HoG features concate-

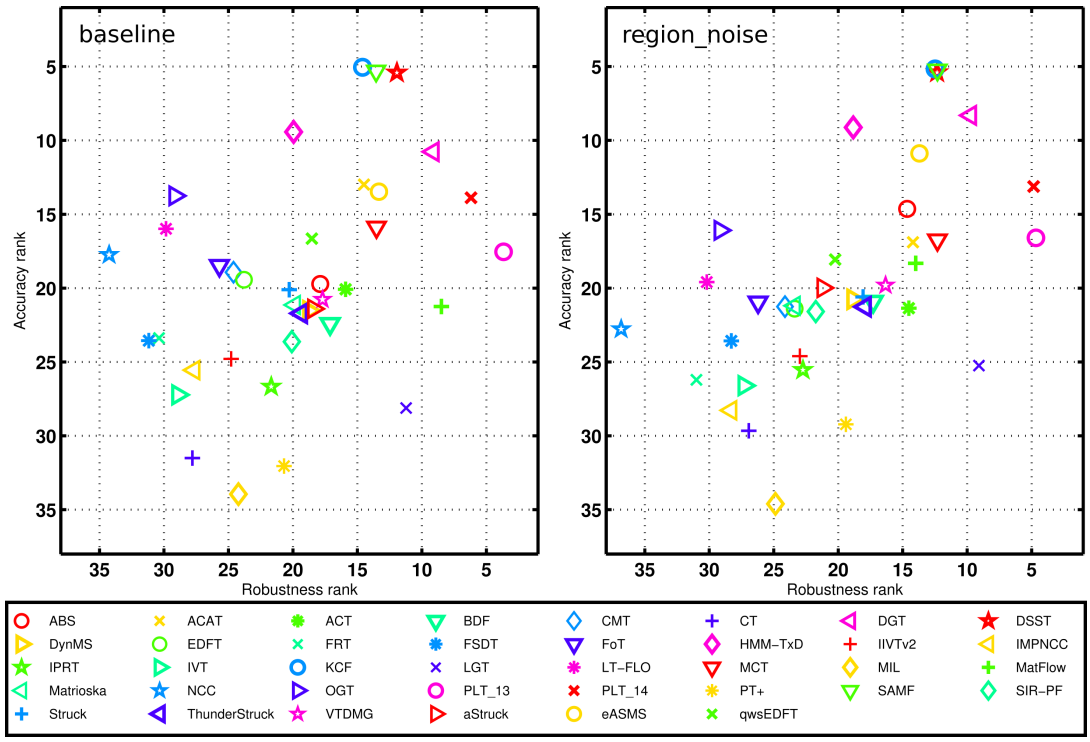


Figure 2.6: Ranking results of VOT 2014 challenge [65]

nated with image intensity values for learning discriminative correlation filters to be used for tracking. Scale estimation, which is the most problematic attribute according to VOT 2013 report, is achieved by a scale-space feature pyramid. The scale and translation filters are kept separate for higher efficiency, i.e., scale is estimated after the optimal translation is found. The resultant translation and scale samples are then used to update scale and translation models.

SAMF [77] is also a kernel correlation filter tracker and it uses scale adaptive feature integration. Similar to DSST, SAMF uses HoG features in collaboration with color naming, a perspective, 11-dimensional color space with mapping in [113]. However, instead of one-dimensional scale filter in DSST, multi-scale adaptation of kernel in SAMF is achieved using a scale pool which is composed of samples with different scales that are interpolated to a fixed size.

KCF, Kernelized Correlation Filter, [47] is developed by the authors of CSK with improvements of multi-scale support, model update and sub-cell peak estimation. As in CSK, training examples are obtained by cyclically shifting a positive example, which is a one dimensional vector and chosen to be HoG fea-

tures in KCF. This shifting operation is shown to correspond to the translation of image patches. The matrix of training samples is then diagonalized using Discrete Fourier Transform (DFT), which enables efficient computation of kernel correlation.

A different approach based on dynamic graphs is used in DGT, Dynamic Graph Tracker by Wen et al. [65]. Graphs are constructed on superpixels and tracking problem is attacked by matching candidate graphs to the target graph using an affinity matrix based on motion, appearance and geometric constraints.

Readers who are interested in the algorithms participated in the challenge, evaluation criteria and/or more detailed ranking results are referred to the challenge report [65].

The most important conclusion that can be deduced from survey papers, benchmarks and challenge reports is the fact that there is a huge variety of possibilities for visual object tracking algorithms. Many different methodologies are experimented by various researches; however, none of them presented a performance that promises to overcome all difficulties in real-world tracking scenarios. Thus, present tracking algorithms have their own capabilities and difficulties. A second very important conclusion is the fact that evaluation results depend heavily on the evaluation metrics. An algorithm that is rated as the best one according to an evaluation metric may be regarded as less successful according to another one. An overview of evaluation metrics used for visual object tracking is given in [105]

CHAPTER 3

DEEP LEARNING - FUNDAMENTALS

3.1 Biological Inspiration

Human brain, which is the main organ of human nervous system, is composed of 100 billion neurons on average and information transmission takes place via 1,000 trillion synaptic connections where neurons are connected to each other [80]. A typical neuron is assembled of a body, called soma, dendrites, which are thin branches connected to the cell body, and a single axon, a special, extra-long filament that is also connected to the soma. A simple illustration of a typical neuron is given in Fig. 3.1. As depicted in the figure, each neuron receives input signal from its feathery dendrites, process the information in its body and transmits the processed signal through the axon, which branches to link neighboring neurons.

Intracellular processes are activated by action potentials, which are electrochemical pulses that emerge from sudden voltage increase across cell membranes above a precise threshold. Generation of action potentials obeys "all-or-none" principle, i.e., they are either generated or not. Larger stimulus does not lead to action potential with larger amplitude and it is the frequency of action potentials that is correlated with the intensity of the stimulus. In addition, a single neuron can get invoked by multiple other neurons, to which it may respond differently. Different input from other neurons may either increase (excite) or decrease (inhibit) the generation of action potentials so the firing rate, i.e., frequency of action potentials, depends on the synaptic connections.

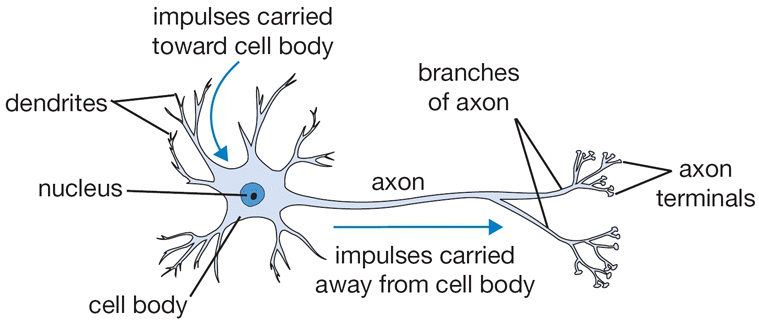


Figure 3.1: Illustration of a typical neuron

Mathematical model of a neuron uses all these principles of human brain. In Artificial Neural Networks (ANN), neurons are computational units that sum a bunch of input after weighting them properly and outputs the value of summation after a non-linearity. Although artificial neuron model is explained in Chapter 3.3, brief explanation of this mathematical modeling is as follows: multiple inputs to an artificial neuron corresponds to synapse connections between multiple neurons. Inputs are weighted that may be regarded as the strength of synaptic connections, and weighted sum is fed to an activation function with a bias, that may model the thresholded behavior of action potentials. The analogy between an artificial neuron and brain computation unit is well depicted in Fig. 3.2.

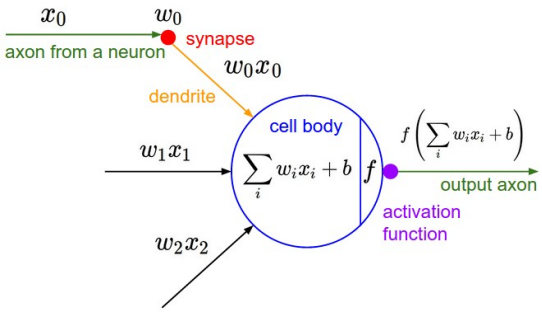


Figure 3.2: Mathematical model of a neuron

Neuron model is not the only way machine learning gets inspired from human physiology. The success of human visual system to extract the relevant information from the huge amount of data it is exposed to gets attention from visual neuroscience community and such a fast and giant computational ability attracts researchers from computer vision and artificial intelligence communities. Human visual cortex, present at the rear of the brain, enables people to recog-

nize patterns by a virtue of complex neuron structure that work parallel in an hierarchical way. The primary visual cortex, which is the best studied area of human brain and also known as V1, receives information from the visual field and transmits it to two primary pathways. Ventral pathways, which is sometimes referred as "what pathway", is responsible for object representation and recognition and follows a path from V1 to inferior temporal cortex through V2 and V4. This path is known to process the information in a sequential manner so that the simple visual forms like edges and corners in V1 are converted into intermediate representations, i.e., feature groups, throughout the way so that high level object descriptions are obtained for decision making in the related area of the brain. This layered structure that interprets visual stimula from retina was the inspiring point for hierarchical representations. An illustration of ventral pathway is given in Fig. 3.3.

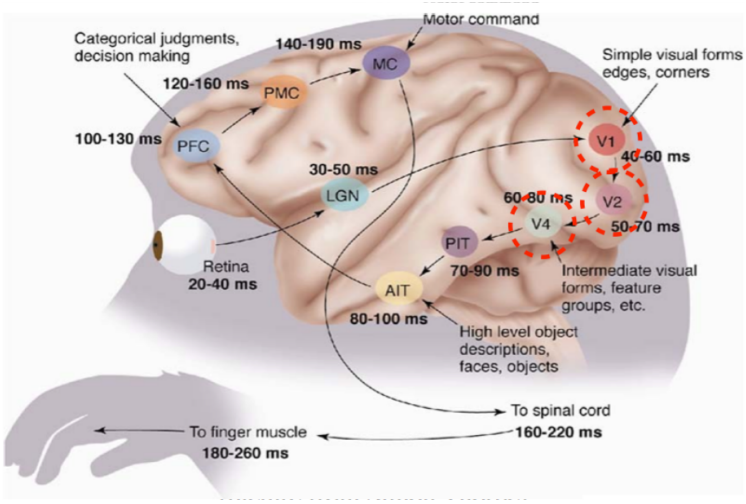


Figure 3.3: Illustration of the ventral pathway of human visual system (image taken from [112])

On the other hand, dorsal pathway is known as "where pathway" and associated with motion and localization.

Construction of complex representations in mammal brain is first revealed by David Hubel and Torsten Wiesel by famous cat experiment in 1959. They placed microelectrodes into the primary visual cortex of an anesthetized cat and presented various stimuli that are composed of light and dark areas to its eyes. What they observed was the rapid firing of some neurons when the light

constitutes a line at a certain angle, while some other neurons fire when the angle is changed. Hubel and Wiesel called these "orientation-sensitive" neurons *simple cells* as they respond to light and dark patterns differently. Another observed type of neurons was what they called *complex cells*, which detected edges of a larger receptive field in a contrast-insensitive manner [54]. They suggested that complex cells receive input from simple cells and this hierarchy of features detection found many applications in machine learning which will be explained in detail later in this chapter.

Despite all this primary inspiration, it is important to emphasize that the research of neural networks aims neither to imitate the human brain nor to generate biologically plausible models. Rather, research mainly focuses on the mathematical properties of artificial neurons to improve the models. The example Yann Lecun, who is one of the inspirational professors in deep learning research, gives in many of his talks and tutorials explain the situation fairly well: he remarks that mankind did not achieve flying by imitating the way birds do it, rather, they understood the basic principles of aerodynamics by observing flying creatures and find convenient ways that are based on the mathematical models of their observations.

3.2 Historical Background

Although it is not possible to survey a research area with such enormous number of publications that increases even more rapidly day-by-day and created its own way of "open-review" in order not to stifle innovation and progress, aim of this part is to overview the evolution of deep artificial neural networks briefly. Readers who are interested in a more detailed survey are highly encouraged to view [101] written by Jurgen Schmidhuber, who intends to assign credit to those who contributed to deep learning in neural networks with over *850* references currently.

This decade has been witnessing the inevitable popularity of deep learning; however, shallow neural network (NN)-like models date back to 1940s. In 1943,

McCulloch and Pitts proposed a non-recurrent neural net that incorporates no learning scheme [81]. They stated a neuron that fires at a certain time provided two conditions: first, none of the neurons should have an inhibitory synapse towards it was firing a time step earlier, and second, more than a certain number of neurons have an excitatory synapse towards it was firing at a time step earlier. Despite the complicated statement and lack of learning scheme, their proposal may be regarded as an early, very simple NN architecture that tries to realize temporal propositional expressions.

On the other hand, in 1949, Donald Hebb presented a learning rule, sometimes referred as "Hebb's rule", by stating that some growth process should take place in two cells if one of them fires the other persistently and repeatedly [45], which actually indicates that neuron model of McCulloch-Pitts should be adaptive according to this biological proposal. In 1958, a major contribution is made with the statement of well-known *perceptron* by Frank Rosenblatt [95] who used McCulloch-Pitts neuron model and Hebb's findings. Perceptron is a single neuron with adjustable synaptic weights and bias and it is built for pattern classification. Although this simplest form of a neural network requires patterns to be linearly separable for classification, it was the first time when a supervised learning scheme incorporates a neural network. In further discussions [96], perceptron is defined to take inputs which are not equally important, i.e., weighted differently, for computation of a thresholding that corresponds to step function. In mathematical form, the neuron model can be defined as in Eq. 3.1 with five inputs, a_1 to a_5 with corresponding weights w_1 to w_5, as in [96]. Weights are initialized randomly and adaption is achieved using binary labeled samples $a^{(j)}, c_a^{(j)}$ iteratively until all the samples are classified correctly with learned weights. Perceptron learning algorithm is summarized in Algorithm 1. Rosenblatt proved the convergence of his learning algorithm in case of linearly separable samples.

$$b = \sum_{i=1}^{5} w_i a_i, \qquad f(b) = \begin{cases} 0 & \text{if } b < \theta \\ 1 & \text{if } b \geq \theta \end{cases} \tag{3.1}$$

Algorithm 1 Perceptron Learning Rule

Random weights $w_i, i = 1, .., N$

Labeled samples $a^{(j)}, c_a^{(j)}, j = 1, .., M$

repeat

Select random $a^{(j)}, c_a^{(j)}$

$error = c_a^{(j)} - f(\sum_i w_i a_i^{(j)})$

for all i **do** $w_i = w_i + learningrate \cdot error \cdot a_i^{(j)}$

end for

until All samples are correctly classified

Perceptron actually emerged from a machine invented in 1957, in which neurons were built as photocells, weights were encoded in potentiometers and update is achieved through electric motors [14]. With an interesting anticipation, they were reported to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence" by New York Times [88]. When they had been shown to be untrainable for many types of patterns, that are basically not linearly separable, loss of interest in perceptrons were as rapid as it attracted attention in artificial intellegence community, so, neural network research experienced a deadlock that lasted years.

After the exploration of simple and complex cells in 1962, Fukushima proposed *neocognitron* [33] in 1979 and employed neurophysiological insights in an artificial neural network that deserves the attribute *deep*. Neocognitron is a hierarchical multilayer neural network that consists of two different type of cells, namely "S-cells" and "C-cells" which are placed alternately. S-cells in the hierarchical structure serves for feature extraction and have variable input connections that are learned during training. Each S-cell responds to a particular feature in its receptive field, which may be local features in lower stages and more global features in higher stages. On the other hand, input connections of C-cells are fixed and these cells are used to handle positional errors. A typical architecture of the neocognitron is depicted in Fig. 3.4. The objective is to recognize the shapes regardless of their locations by depending on the geometrical symmetry [34] and this trainable network depends on the self-organization for learning in an un-

supervised manner. Thus, neocognitron is very similar to famous convolutional neural networks in terms of objective and architecture but lacks the notion of supervised training with backpropagation for weight updates.

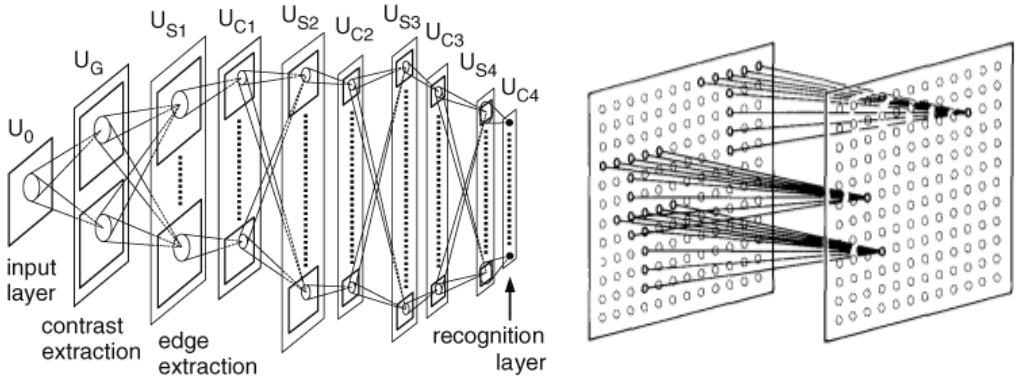


Figure 3.4: (a) A typical architecture of the neocognitron [32] and (b) input interconnections to the cells within a single cell-plane (figures takes from [34])

Efficient error backpropagation for arbitrary NN-like networks with possible sparse connections is first proposed in a master's thesis in 1970 [78]. First NN-specific application of backpropagation is reported to be in 1981 in the survey paper of Schmidhuber and related work is published in 1986 [71]. Use of backpropagation for neural networks gained popularity with the work of Rumelhart et al. [99] that presented useful representations in hidden layers. However, research was focused on networks with few layers since additional hidden layers were believed to offer no more benefits compared to the shallow ones due to the difficulties in training. In 1989, LeCun et al. applied backpropagation to the hand-written digit recognition problem via a convolutional neural network with adaptive connections and weight-sharing, and introduced the very famous data set of hand-written digits, MNIST [73]. Proposed networks employed max-pooling and training was sped up using graphic cards, which are basic and well-known ingredients of today's competition-winning deep architectures.

The problem of vanishing gradients, which is a phenomenon that explains shrinking or exploding cumulative backpropagated errors due to saturating activation functions, is first investigated in 1991 in a diploma thesis [52]. Later, much of the research focused on this fundamental problem of deep learning which makes deep networks hard to train and several solution proposals have been presented.

The idea of unsupervised pre-training that greatly facilitates supervised credit assignment either by stacked *autoencoders* or *Deep Belief Networks* was one of the breakthroughs in deep-learning research [48]. Propositions that incorporate Hessian-free optimizations are also known to alleviate the problem for both feed-forward and recurrent networks. Introduction of new labeled datasets that consist of millions of images and the computation power of today's graphics processing units (GPUs) also contributed to deep learning research.

Although machine learning research focused on other methods like Support Vector Machines (SVMs) more during 2000s, results of a convolutional neural network on *ImageNet* classification benchmark drew the attention back to neural networks. Proposed network, which is called as "AlexNet" today, achieved top-5 error rate of 15.3% on the classification task of Large Scale Visual Recognition Challenge (ILSVRC) that targets 1000 classes of objects on images of size 256x256 [67]. The success of a deep feed-forward network on such a well-known challenge, where the second best result was 26.2%, has influenced the extensive research on deep learning by both machine learning and computer vision communities since then.

3.3 Feed-Forward Networks

3.3.1 Artificial Neuron

As stated earlier in this chapter, the basic unit of a artificial neural network is a computational unit which is called as neuron. A single artificial neuron performs a particular computation of the input, which may be decomposed into two steps called input activation (pre-activation) and output activation (neuron activation). Input activation refers to summing the input with a *bias* with probably unequal importances (*weights*) and output activation feeds the weighted sum to an activation function and dictates the output of the neuron accordingly. If we denote the input of d dimensions as $\mathbf{x} = [x_1, ..., x_d]^T$, weights as w_i, bias as b and activation function as $g(.)$, pre-activation and activation functions, $a(\mathbf{x})$ and $h(\mathbf{x})$, are computed as given in Eq. 3.3

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^T \mathbf{x} \tag{3.2}$$

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g\left(b + \sum_i w_i x_i\right) \tag{3.3}$$

Note that a single neuron may be regarded as a binary classifier when the activation function is the unit step function, which corresponds to the perceptron. However, there are many different activation functions used for neural networks each with its own offerings.

- One option is the linear activation, that does not squash input and does not have any lower or upper bounds. Linear activations are usually not used in multilayer neural networks since cascaded linear layers also indicates a linear operation, which in fact can be represented with a single linear layer.
- Sigmoid, $sigm(.)$, is a commonly used option for the activation function and squashes the neuron's pre-activation between 0 and 1.
- Hyperbolic tangent, $tanh(.)$, is another strictly increasing option with slightly different characteristics and different lower and upper bounds, -1 and 1, so that the output is centered at zero.
- Rectified linear activation functions, $reclin$, that has become popular due to the sparsity it introduces to neuron activities.

Explicit definitions of the last three activation functions are given in Eq. 3.6 and all of the given functions are illustrated in Fig. 3.5.

$$sigm(a) = \frac{1}{1 + \exp(-x)} \tag{3.4}$$

$$tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1} \tag{3.5}$$

$$reclin(a) = max(0, a) \tag{3.6}$$

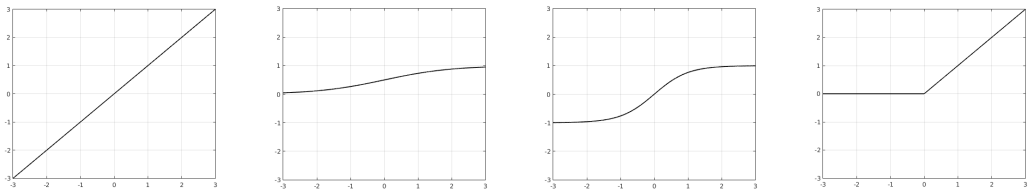


Figure 3.5: Different activation functions: (a) linear (b) sigmoid (c) hyperbolic tangent (d) rectified linear

Although these would be more clear later in this chapter, there are a few things to note on the activation functions. Despite the frequent use in neural network history, sigmoids have fallen out of favor since they cause the problem of vanishing gradients. If the local gradient is small in a hidden layer, i.e., neuron is saturated and takes the values of sigmoid on its tails, errors are backpropagated to lower layers after a multiplication with a very small number. Different initialization strategies for the weights are introduced to overcome the problem of vanishing gradients when sigmoid or hyperbolic tangent activations are used; however, other type of activation functions that do not saturate, such as rectified linear activations, are more commonly used nowadays. Rectification may also cause the gradients to "die" in case of improper initialization and learning rate choice. Leaky rectified units, which does not output zero but rather dictates a small portion of the input when it is less than zero, are proposed to overcome dead gradient problem.

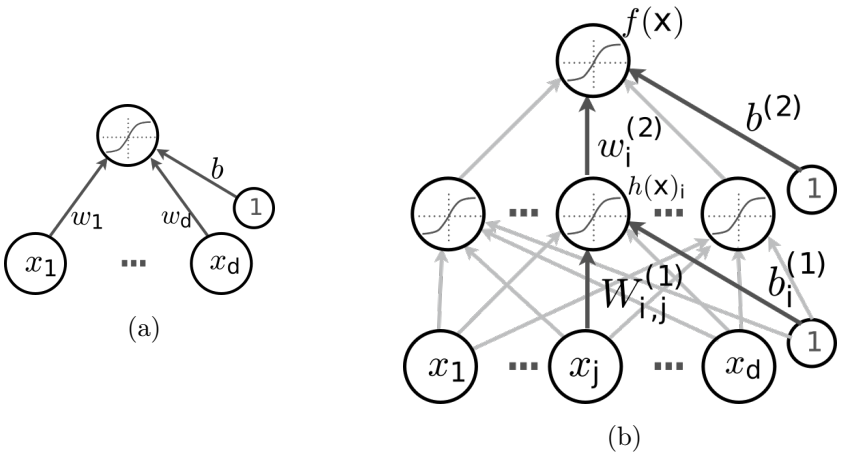


Figure 3.6: (a) An artificial neuron and (b) multi-layer feed-forward neural network with one hidden layer (images taken from [70])

3.3.2 Neural Networks

A single neuron may serve as a binary classifier as stated earlier, and logistic regression is the specific case when sigmoid activation function is used for classification at the output layer. However, decision boundary of such a classifier is linear and many real world problems employ classification tasks of data that are not linearly separable.

Multiple neurons are used to construct networks in order to increase the *capacity*, which refers to the space of representable functions. In other words, for the success of a linear classifier, it is necessary to transform input into a new representation where data is linearly separable and utilization of multiple neurons attacks the problem of finding a proper representation. Different structures are possible according to the type of connections in the network, for example, feed-forward networks allow information to travel only in one direction but recurrent networks employ feedback loops so that information may travel in both directions. Thus, connections of neurons may be represented as an acyclic graph for feed-forward neural networks and cyclic graph for recurrent ones. Feed-forward networks are often organized into distinct layers and named differently according to the connections between neurons. If each pair of neurons in adjacent layers are connected, network is called to be fully-connected and the layers in the middle of the network, i.e., that are not at the top or bottom, are called as hidden layers. Neurons connected to the input are named as input layer and there usually exists an output layer on top of hidden layers for classification. A single neuron and a multilayer neural network with one hidden layer are illustrated in Fig. 3.6.

3.3.3 Capacity and Overfitting

Feed-forward neural networks with a single hidden layer and linear output is stated to approximate any continuous function with a finite error provided enough hidden units [22], which is known as universal approximation theorem [53] Fig. 3.7 shows the capacity of a single neuron and illustrates its bi-

nary classification power for a two-dimensional input vector. A neural network that contains one hidden layer with four neurons, which may generate a simple, bumpy output of the input vector is also given in Fig. 3.7. This illustration gives an intuition of universal approximation problem.

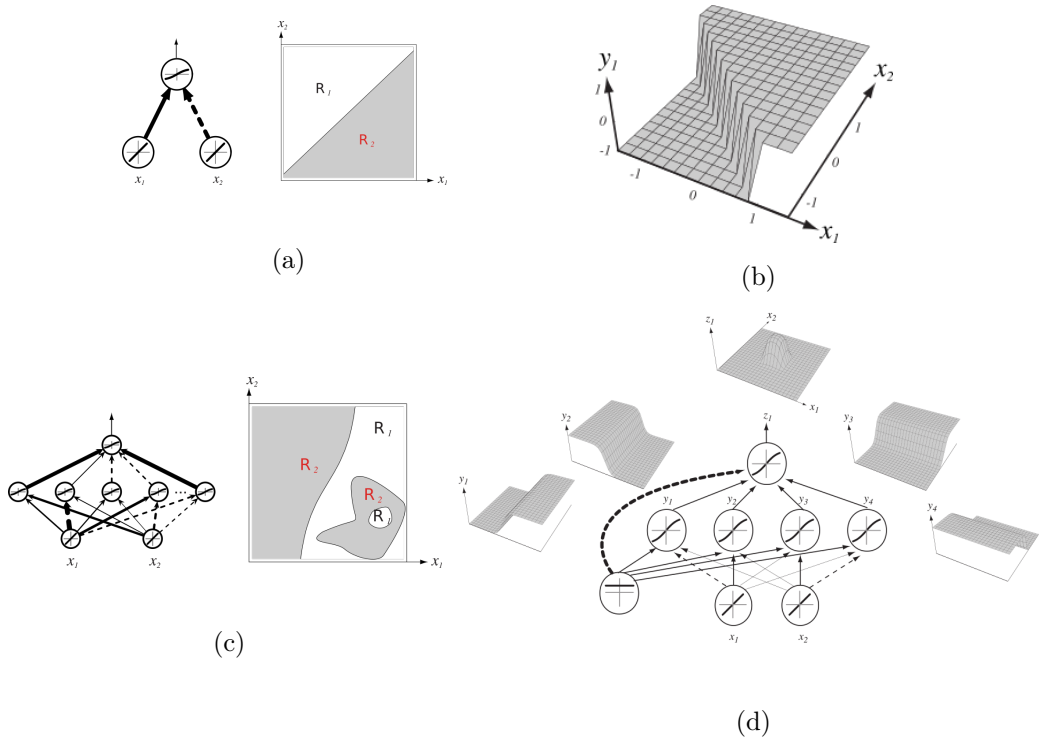


Figure 3.7: A single neuron illustrated in (a) is capable of binary classification, which is depicted in (b) for two-dimensional data. Multilayer neural networks with hidden layers may achieve classification of data that are not linearly separable (c) and approximate continuous functions arbitrarily well (d) (images taken from [115])

Although neural networks with a single hidden layer are shown to approximate any continuous function arbitrarily well, parameters of neural networks are never guaranteed to converge to the specific values of aforementioned approximation. The more the parameters of a network for a specific problem, the harder the training algorithm will converge to a "good" local minimum in most of the cases. A well-known fact is that functions represented compactly with k layers may require exponential size with k-1 layers, which is proved for logic gates [42], formal neurons [55] and radial basis function (RBF) units [13]. In other words, functions that are compactly representable with a deep architecture would require exponentially more units, and therefore parameters, when represented with a shallow

network. Thus, compact representations introduced by *deep* feed-forward neural networks is a key point for better-trained parameters and *better* representation, provided that the vanishing gradients problem is overcome with a proper training strategy.

Before delving into training strategies for deep architectures, there are some important issues to be emphasized about the capacity of neural networks. Representational power is shown to increase with additional layers of neural networks and higher number of neurons, but is it always true that the higher the number of hidden layers and/or neurons the better the representation is? This question brings us to the problem of over-fitting, which may be well-explained by a demonstration prepared by Andrej Karpathy [61]. Suppose that the problem is the classification of two-dimensional data. In Fig. 3.8, performance of different neural network architectures is shown to understand the effect of hidden layer number and sizes. Results in the first row corresponds to a fully-connected feed-forward neural network with a single layer of different number of neurons. The first and most obvious observation is the fact that network learns to classify more complicated data as the number of neurons in the network is increased. However, the problem is that it starts to *memorize* the data which probably contain noise in real world problems. This phenomenon of perfectly fitting to the present dataset instead of learning the underlying relationship is known as *overfitting*. Another example of overfitting is presented in the second row of Fig. 3.8, which illustrates the classification results of networks with different number of hidden layers of fixed number of neurons. What we expect from neural networks is not to fit the model to all seen examples, as done by the networks with higher number of neurons and hidden layers that tries to classify even outliers, rather, find a network that generalizes so that it performs well not only on the training data but also on the test data, which contains unseen examples.

3.3.4 Optimization

This simple example emphasized the importance of architecture choice; however, choosing the *right* architecture is not the only way to avoid overfitting.

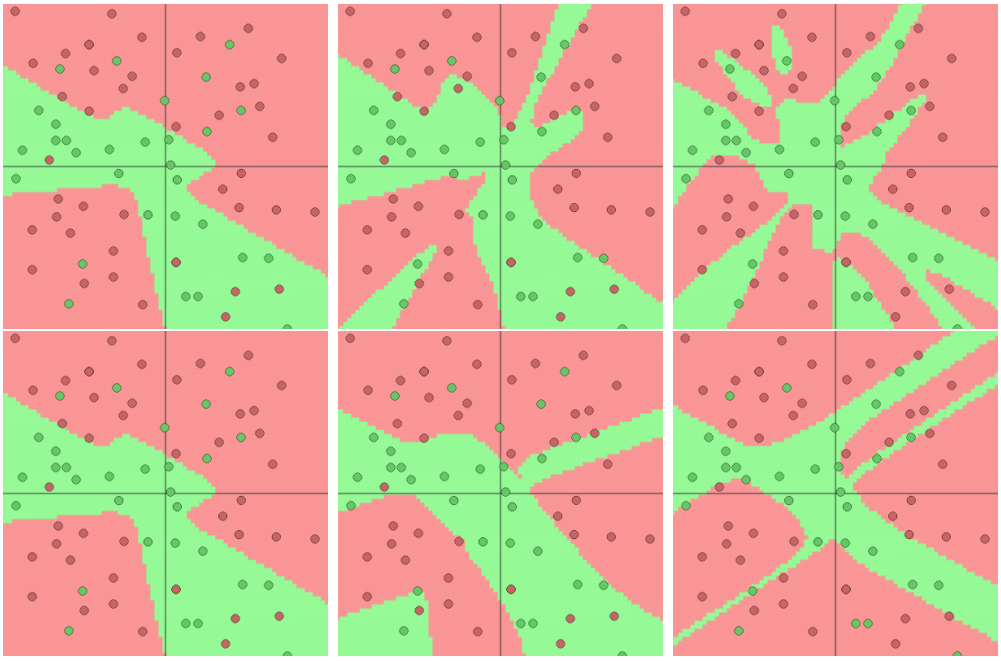


Figure 3.8: Toy example of classifying two dimensional data points with a neural network of different architectures. Images in the first row presents the results of neural networks with a single layer of 3,6 and 20 neurons, from left to right. Images in the second row is the classification results of neural networks with 1,2 and 3 hidden layers of 3 neurons. As the number of neurons/hidden layers increased in a neural network, it tries to fit the data perfectly, which probably does not generalize to the test data.

Training strategy is as important as the network architecture. The main principle behind training neural networks is known as empirical risk minimization and the objective is to learn weights and biases for classification in a supervised manner, i.e., when data is available with target labels that should be predicted by the designed neural network. Training of a neural network is regarded as an optimization problem where a target cost function of parameters $\theta = \{\mathbf{W}, b\}$ are minimized by proper optimization methods. A general framework for a learning algorithm employs a loss function $l(.)$ that compares the output of the neural network and the target label, a labeled data set composed of input $\mathbf{x}^{(t)}$ and corresponding label $y^{(t)}$ for $t = 1, .., T$, a regularizer $\Omega(.)$ that penalizes certain values of θ which will avoid overfitting and aims generalization in most cases, and an optimization procedure with an initialization method. Hence the overall framework can be summarized as in Eq. 3.7 where $f(x^{(t)}; \theta)$ resembles the output of the neural network for input $x^{(t)}$.

$$\arg\min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta) \tag{3.7}$$

Stochastic Gradient Descent One very common exercise for the minimization of any function is to decrease the parameters in the opposite direction of the gradient, which is known as gradient descent. In case of neural networks, there may be tens or hundreds of thousands of images (remember that in ImageNet, it is 1.2 million training images of size 256x256), it would be exhausting to compute the loss function and/or its gradient for all of the input samples. Therefore, it is very common to divide the training set into subsets called batches, over which gradients are computed and parameters are updated accordingly. Parameter update is realized for all batches in the training set so that every sample is visited, and this iteration of all samples is called one "epoch", whereas the procedure is named as (mini-)batch gradient descent. Another practical variant of gradient descent is stochastic gradient descent that enables on-line optimization by considering only one sample for each iteration of parameter update. However, stochastic gradient descent is usually utilized to indicate parameter updates with mini-batches and the reason for such an approximation with batches is the fact that the input data is usually correlated and the average of gradients over a subset with reasonable number of elements is believed to lead to a good direction for the minimization process. The number of samples in a batch is a *hyperparameter* as well as the learning rate, α, which is the amount of movement, step size, in the opposite direction of gradient. These hyperparameters are determined using a validation set. The stochastic gradient descent algorithm is described in Algorithm 2 and it is usually utilized with batches instead of samples.

Algorithm 2 Stochastic Gradient Descent

for N iterations **do**
  for each training sample $(x^{(t)}, y^{(t)})$ **do**
    $\Delta = -\nabla_{\theta} l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$
    $\theta \leftarrow \theta + \alpha \Delta$
  end for
end for

When the task is classification, neural networks are trained to estimate the class probabilities given the input data, $f(x)_c = p(y = c|\mathbf{x})$, which should be maximized. This maximization problem is converted to a minimization problem using negative log-likelihood. An example loss function for classification task is given in Equation 3.8 where 1 represents the identity function.

$$l(f(x;\theta), y) = -\sum_c 1_{(y=c)} \log f(x)_c = -\log f(x)_y \tag{3.8}$$

For multi-class classification task, categorical probability densities can be found by exponential normalization, known as soft-max. The soft-max loss function for class c_i over a total of C classes is given in Eq. 3.9.

$$l(f(x;\theta), y)_{c_i} = -\log\left(\frac{\exp f(x)_{c_i}}{\sum_{c=1}^{C} \exp f(x)_c}\right) \tag{3.9}$$

Backpropagation Once the loss function is set, parameters in every layer should be updated according to the gradient by the amount of learning rate. For this purpose, errors are required to propagate from the output layer up to the input layer, which is known as *backpropagation*. Regarding the network in Fig. 3.6b with the loss function in Eq. 3.8, following derivatives are involved in the backpropagation:

- Partial derivative w.r.t. output pre-activation, $\frac{\partial(-\log f(x)_y)}{\partial a^{(L+1)}(x)_c}$
- Partial derivatives w.r.t. hidden layer activations, $\frac{\partial(-\log f(x)_y)}{\partial h^{(k)}(x)_j}$
- Partial derivatives w.r.t. hidden layer pre-activations, $\frac{\partial(-\log f(x)_y)}{\partial a^{(k)}(x)_j}$
- Partial derivatives w.r.t. parameters , $\frac{\partial(-\log f(x)_y)}{\partial \mathbf{W}_{i,j}^{(k)}}$ and $\frac{\partial -\log f(x)_y}{\partial b^{(k)}}$

Derivation of all these partial derivatives are out of this thesis' scope; however, resultant expressions for gradients are given in Eq. 3.10 for the sake of completeness. Loss function $-\log f(x)_y$ is denoted as $\mathcal{L}$ for easier understanding. Readers who are interested in the derivation process are encouraged to check the online course materials by Hugo Larochelle [68].

$$
\begin{aligned}
\nabla_{f(x)}\mathcal{L} &= \frac{-1}{f(x)_y}\left[1_{(y=0)}, ..., 1_{(y=C-1)}\right] \\
\nabla_{h^{(k)}(x)}\mathcal{L} &= \mathbf{W}^{(k+1)^T}\left(\nabla_{a^{(k+1)}(x)} - \log f(x)_y\right), \\
\nabla_{a^{(k)}(x)}\mathcal{L} &= \left(\nabla_{h^{(k)}(x)} - \log f(x)_y\right) \odot \left[..., g'(a^{(k)}(x)_j), ...\right] \\
\nabla_{\mathbf{W}^{(k)}}\mathcal{L} &= \left(\nabla_{a^{(k)}(x)} - \log f(x)_y\right) h^{(k-1)}(x)^T, \\
\nabla_{b^{(k)}}\mathcal{L} &= \nabla_{a^{(k)}(x)} - \log f(x)_y
\end{aligned}
$$

Equations show that backpropagation algorithm requires the values of activation and preactivation for gradient calculation; in other words, it assumes forward propagation. Once the forward propagation is realized, backpropagation algorithm may update gradients according to Algorithm 3

Algorithm 3 Backpropagation

Forward propagate the input $\mathbf{x}$ and compute $f(x)_y$

Compute output gradient: $\nabla_{a^{(L+1)}(x)} - \log f(x)_y$

for k from L+1 to 1 **do**

Compute gradients of hidden layer parameters: $\nabla_{\mathbf{W}^{(k)}} - \log f(x)_y, \nabla_{b^{(k)}} - \log f(x)_y$

Compute gradient of hidden layer below: $\nabla_{h^{(k-1)}(x)} - \log f(x)_y$

Compute gradient of hidden layer (pre-activation) below: $\nabla_{a^{(k-1)}(x)} - \log f(x)_y$

end for

Regularization As emphasized earlier, there are different ways of controlling the capacity of a neural network so that overfitting is avoided. One option is to regularize the parameters of the network, which is employed by an additional term in the loss function to be minimized. Most common form of regularization is $L2$ regularization, which makes the regularization term in the cost function $\Omega(\theta) = \frac{1}{2}\sum_k \sum_i \sum_j \left(\mathbf{W}_{i,j}^{(k)}\right)^2$, and shrinks weights with a square penalty. Networks with such a regularization are encouraged to use all of the inputs instead of using some of them more often. Small weights are interpreted in such a way that they do not learn the local noise in data since the alteration of a few ran-

dom inputs would create a negligible change. Result of weight regularization is a simpler network compared to a network with large weighs that are adapted perfectly to the noise in the data.

Another option is to employ a $L1$ regularization, for which the regularization term $\Omega(\theta) = \sum_k \sum_i \sum_j |\mathbf{W}_{i,j}^{(k)}|$ forces some weights to be zero and thus introduces sparsity. The reason why $L2$ regularization only shrinks weights towards zero whereas $L1$ regularization usually results in sparse weights is the fact that the shrink amount in $L2$ is proportional to the weights due to partial derivate; however, weights shrink by a constant amount, either -1 or 1 according to their sign, in $L1$ regularization. Regularization, either $L1$ or $L2$, is used for weights, not biases.

There are also some works that normalize weights after each epoch instead of introducing an additional term to the loss function.

Recently, a new regularization method called dropout is introduced [107] which has presented a lot of successful results in deep learning. Dropout simply proposes discarding some neurons and therefore not updating related parameters during training with probability p. Forward propagation and backpropagation of errors are carried out only through the neurons chosen randomly in each epoch and related parameters are updated so that complex co-adaptation of neurons are claimed to be prevented. One important thing is that dropout is applied during training and never utilized once the parameters are learned.

Learning Rate Optimization scheme is also very important since the loss function is believed to be full of local minima and plateaus, and neural network research intends to find a *good* minimum which leads to good representation for the specific task. Effect of two important aspects in the optimization scheme has been deeply investigated: the choice of learning rate and update rule.

In terms of learning rate, it is a well-known fact that small learning rate results in slow convergence whereas large learning rate may cause parameter vector to bounce around and hardly settle down. In some optimization problems, instead of trying to choose a proper learning rate, Newton's method is utilized in order

to consider the curvature of the cost surface for an efficient update. However, Newton's method requires the computation of the inverse of Hessian matrix, the square matrix of second partial derivatives, which is possible if the cost function is at least locally convex. This computationally expensive method is preferred when the number of parameters are not large. The update rule with Newton's method is given in Eq. 3.11 where $H(f(\mathbf{x};\theta))$ is the Hessian matrix.

$$\theta_{t+1} = \theta_t - (\nabla^2_{\theta_t} l(f(\mathbf{x};\theta_t), y))^{-1}(\nabla_{\theta_t} l(f(x;\theta_t), y)) \tag{3.10}$$

$$= \theta_t - [H(f(\mathbf{x};\theta_t))]^{-1}(\nabla_{\theta_t} l(f(x;\theta_t), y)) \tag{3.11}$$

It is also possible to anneal the learning rate throughout the training procedure to promote generalization. In addition, stochastic gradient descent is not guaranteed to converge when the learning rate is constant. In fact, the constraints that will lead stochastic gradient descent to converge are $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$. Therefore, different decay methods are employed for annealing the learning rate. One option is to decrease the learning rate by some amount every (few) epoch(s), which may also be realized only if the validation error stops improving. Other possibilities are multiplying the learning rate by a constant less than 1, using exponential decay or time dependent decay with a formula like $\alpha = \alpha_0/(1 + kt)$ where α_0 is the initial learning rate and k is a hyperparameter, or updating the learning rule as $\alpha \leftarrow \alpha/t^{\delta}$ where δ is a hyperparameter. It is a common practice to use a constant learning rate for few updates and decrease it to avoid overfitting by using any of these rules. Fig. 3.9 illustrates the convergence of a loss minimization process for different learning rates and reveals the importance of a proper selection.

Update Rule When it comes to update rule, different approaches, such as momentum update, are proposed the accelerate the convergence of minimization process. In vanilla gradient descent, which is the regular policy, parameters θ are updated according to the rule in Eq. 3.12, where Δ is the gradient and α is the learning rate.

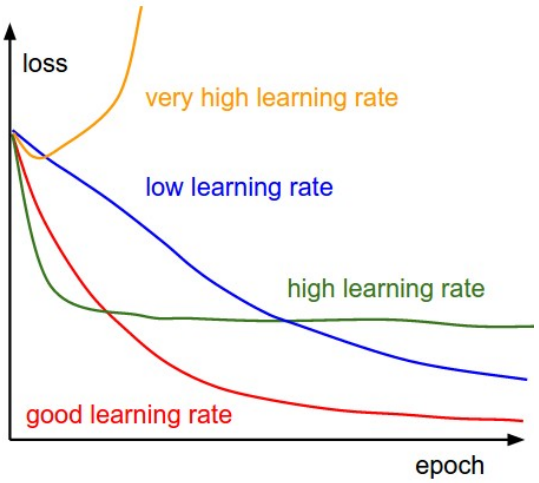


Figure 3.9: Illustration of different learning rates for the loss minimization process

$$\theta_{t+1} = \theta_t - \alpha \nabla \mathcal{L}(\mathbf{x}; \theta_t) \tag{3.12}$$

The point of the momentum update is to increase the speed of convergence without heavy computations as in Newton's method, simply by making use of the previous gradient updates. This method introduces a notion of "velocity" which resembles the weighted accumulation of gradients in previous epochs, thus, acts as a friction to gradually decrease the kinetic energy of the optimization system. Momentum update rule is given in Eq. 3.14, where v is used for velocity vector which is initialized to zero and μ is the momentum coefficient. Similar to learning rate, momentum coefficient can be annealed in such a way that it has a low value in the beginning of the learning in order to allow free movement but annealed to higher values when procedure probably finds a good path for the minimization of the loss function.

$$v_{t+1} = \mu v_t - \alpha \nabla \mathcal{L}(\mathbf{x}; \theta_t) \tag{3.13}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.14}$$

Nesterov's Accelerated Gradient (NAG) [87] is also a first-order optimization method with a better convergence rate than gradient descent. The key difference

between the classical momentum method in Eq. 3.14 and NAG in Eq. 3.16 is the order of application of momentum and velocity updates. In Eq. 3.14, gradient is computed before velocity update whereas Nesterov proposes the other way, i.e., gradient is computed on weights after the addition of momentum, which seems to change v in a quicker and more responsive way [108]. However, it should be noted that convergence theories for momentum update methods considers an environment without noise and they do not retain asymptotic local rate of convergence in stochastic settings.

$$v_{t+1} = \mu v_t - \alpha \nabla_\theta \mathcal{L}(\mathbf{x}; (\theta_t + \mu v_t)) \tag{3.15}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.16}$$

AdaGrad [27], an adaptive learning rate method, proposes higher step sizes for rarely seen updates whereas the effective learning rate of high gradients is decreased. Method makes use of the update information of previous iterations and formulated in Eq. 3.17.

$$\theta_{t+1} = \theta_t - \alpha \frac{\nabla_\theta \mathcal{L}(\mathbf{x}; \theta_t)}{\sqrt{\sum_{t'=1}^{t} \left(\nabla_\theta^2 \mathcal{L}(\mathbf{x}; \theta_{t'})\right)^2}} \tag{3.17}$$

RMSprop is an unpublished yet known method that proposes keeping a moving average of squared gradients for weight updates, which is given in Eq. 3.19. γ in the equation stands for the decay rate. AdaDelta [125] is another variation of AdaGrad that also employs the moving average of squared gradients over a window with an additional correction step that is based on Hessian approximation.

$$rms_{t+1} = \gamma rms_t + (1-\gamma) \nabla_\theta^2 \mathcal{L}(\mathbf{x}; \theta_{t+1}) \tag{3.18}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\nabla_\theta \mathcal{L}(\mathbf{x}; \theta_{t+1})}{\sqrt{rms_{t+1}}} \tag{3.19}$$

Initialization Initialization of the weights can also affect the learning process. As mentioned earlier, saturation of neurons, that is neurons' taking values in the tails of the activation function, results in small partial derivatives and therefore vanishing gradient problem. On the other hand, all-zero initialization is not rational since all neurons would behave the same way with the same initial points. As a solution, biases are initialized as zero and weights are initialized in such a way that all neurons are random and close to zero at the beginning, which is expected to result in distinct updates and proper learning. A common heuristic is to normalize the initial parameters according to the number of inputs.

Determination of Hyperparameters Variety of approaches to the minimization problem reveals the importance of training process in deep learning algorithms. Determining a cost function for the objective problem is rather easy compared to the selection of architecture, optimization method and determination of hyperparameters. Hyperparameters are also selected via different methods and two very common choices are using grid search for the selection of best hyperparameters in the related space or searching them randomly.

Early Stopping The process of training itself is important since it will directly affect the performance of the neural network for the given problem. If possible, observing the loss function change over a validation set is a good option to avoid overfitting and examine the convergence properties. Another hyperparameter, number of epochs, is usually interfered by stopping the optimization process when the loss or classification error over the validation set starts to increase. This interference is known as *early stopping* and a case that illustrates the evolution of loss over training and validations sets is depicted in Fig. 3.10.

Bias-Variance Trade-off If the minimization process is not interfered in by checking the training error over the validation set, the difference of loss over training and validation sets in the following epochs will probably diverge, which is a strong indication of overfitting. The difference between training and validation error is an example of high variance, when overfitting phenomena is

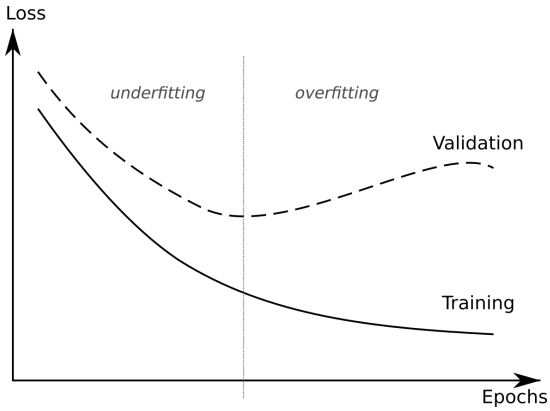


Figure 3.10: Evolution of loss over training and validation sets. Iterations less than required for good convergence will result in a model that could not fit the data. On the other hand, allowing model to iterate more than necessary will cause to memorization of data so that model can not generalize well to new (test) data.

investigated within bias-variance trade-off. Remember that overfitting refers to models that fits perfectly to available training data and cannot generalize well to new test data. Within the bias-variance trade-off, generalization error can be interpreted as the sum of (squared) bias and variance, where variance refers to the sensitivity of the model to the changes in the training set and bias is usually used to indicate the dissimilarity between the trained model and the true one. For a better understanding, different bias-variance cases are illustrated in Fig. 3.11. f^* in the cartoon figure stands for the true, target model and gray areas depicted as f resembles the neural networks that can be trained using different training sets. Our ultimate aim is to find a good trade-off between bias and variance with proper selection of architecture, hyperparameters, update method and regularization method.

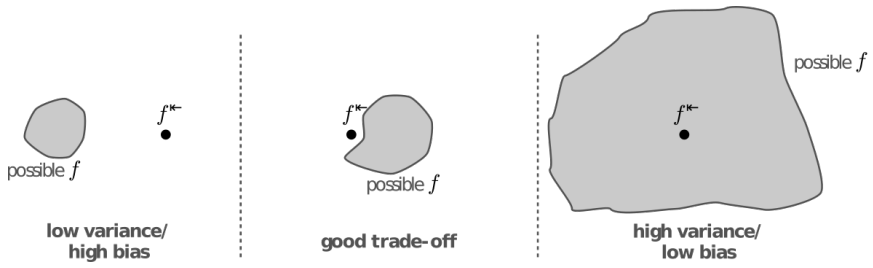


Figure 3.11: Illustration of bias-variance trade-off for neural networks

3.4 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a type of feed-forward networks explained in Part 3.3 with some distinguishing properties. Main objective of CNN architecture is to exploit the 2D topology in images in a hierarchical way, which is very similar to the working principal of human visual cortex. Local connectivity, parameter sharing and subsampling hidden units bring invariance to certain variations and ability to handle larger images compared to fully-connected feed-forward neural networks.

Local connectivity, as can be understood from the name, is the proposition that each hidden unit in a neural network is connected to a small number of units in the previous layer that are spatially localized. However, hidden units are connected to all channels of the input, i.e., connectivity is local in space but full along the input depth.

Parameter sharing refers to sharing of parameters across the neurons of the same channel, or so-called *feature maps*, which leads to extraction of same features at every position. Feature maps are obtained by convolving set of parameters, sometimes referred as filters or kernels, with the image. If k_{ij} is the j^{th} convolution kernel for i^{th} input channel x_i, i.e., k_{ij} is the hidden weight matrix W_{ij} with its rows and column flipped, the j^{th} activation function y_j is computed as in Eq. 3.20, where $g(.)$ is the activation function and operator $*$ in Eq. 3.21 resembles the discrete convolution operation.

$$y_j = g\left(b + \sum_i k_{ij} * x_i\right) \tag{3.20}$$

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q} \tag{3.21}$$

Both local connectivity and parameter sharing decrease the number of parameters to be trained by a great amount. Suppose that we have an image of size [227 x 227 x 3], which is the case for ImageNet dataset. If we had a fully-connected layer where parameters are not shared, there would be 227*227*4 = 157,587

parameters, i.e., weights and biases, for each neuron in the upper hidden layer. If the weights are locally connected and we use a kernel size of 11, which is sometimes referred as *receptive field*, there would be number of 11x11 parameters for each channel for each neuron, which results in 11*11*4*(227-11+1)*(227-11+1) for each feature map in the upper hidden layer where the number of neurons are set to (227-11+1)*(227-11+1) as a result of convolution operation.

Discrete convolution is sometimes not evaluated for each pixel in the input image, rather, the neurons in the upper layer are set to S spatial units apart, where S is called as stride. For example, if the stride is equal to 4, we evaluate the convolution for 4 pixels apart, thus there would be ((227-11)/4 + 1)*((227-11)/4 + 1) neurons for each resultant feature map. As a result, introducing stride of 4 pixels decreases the number of parameters by a factor of 16.

Finally, if parameters are shared among neurons, only 11*11*4 parameters are learned for each feature map in the upper layer. Such a decrease in the number of parameters are compensated with higher number of feature maps in order to increase the variety of features to be learned.

The conclusion of this many multiplicative calculations is actually rather simple, weight sharing and local connectivity properties of convolutional neural networks help keeping the number of parameters manageable for real-world data. Properties of local connectivity and weight sharing are further illustrated in Fig. 3.12.

Although there are some exceptions, convolutional neural networks are generally composed of cascaded convolutional layers followed by periodical pooling layers. Pooling layers reduce the spatial size, and hence number of parameters, by resizing each feature map individually using a maximum or averaging operation. Downsampling realized by pooling layers may be interpreted as a simplification of the information in the output of convolutional layers, i.e., pooling layers determine if a feature exists within a window regardless of its position.

Some convolutional architectures also employ normalization layers in order not to get affected from contrast differences. An example sub-block that incorpo-

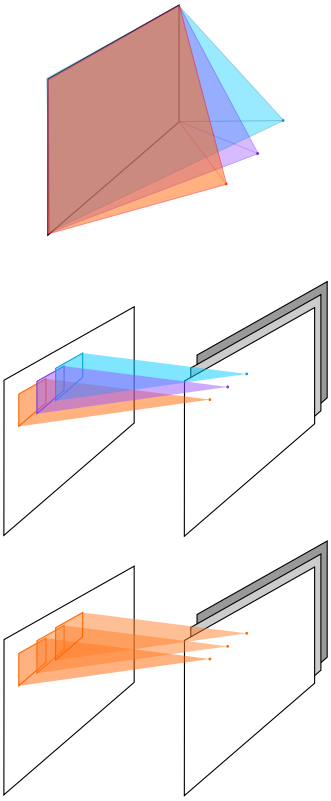


Figure 3.12: Illustration of (a) fully-connected layers (b) local connections and (c) weight sharing (images taken from [7])

rates local normalization is illustrated in Fig. 3.13. In addition, input data of convolutional neural networks is sometimes preprocessed by practicing mean-subtraction, dimension reduction via PCA and/or whitening.

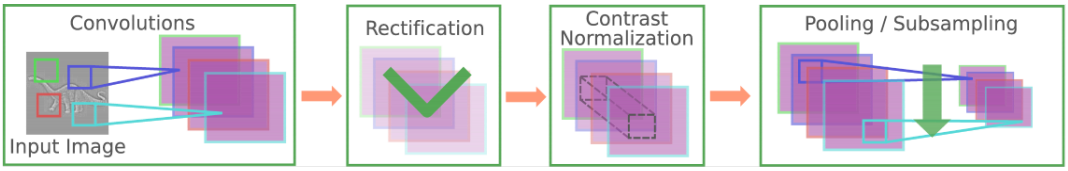


Figure 3.13: A frequent subblock of deep convolutional neural networks

In summary, small-sized filters of convolutional neural network layers are extended through the full depth of the previous layer, they are shared across neurons and the aim is to extract the same features at every position. A single feature map of a layer is generated by sliding a single filter whose depth is equal to the depth of input and realizing a discrete convolution. Multiple filters are used in a hidden layer in order to capture different spatial characteristics and output volume is formed by stacking the filters along depth, which

is sometimes called as "filter bank". Convolutional layers are usually followed by pooling layers and non-linearity functions. Successive application of convolutional and pooling layers with non-linearities result in spatially downsampled outputs which extends through its depth as a virtue of multiple filters in convolutional layers. Resultant output volume is usually processed by fully-connected layers at the highest levels of the architeture, where the very last layer, i.e., output loss function, depends heavily on the task.

A typical convolutional architecture, LeNet-5 [74] is one of the first attempts to apply a neural network to a real world problem such as document recognition. Proposed architecture is composed of convolutional layers followed by subsampling, the result of which fully-connected to RBF units after hyperbolic tangent function. In contrast to more recent architectures, proposed method uses convolutional neural networks to obtain codes to be classified by Gaussian connections. The overall network is illustrated in Fig. 3.14.

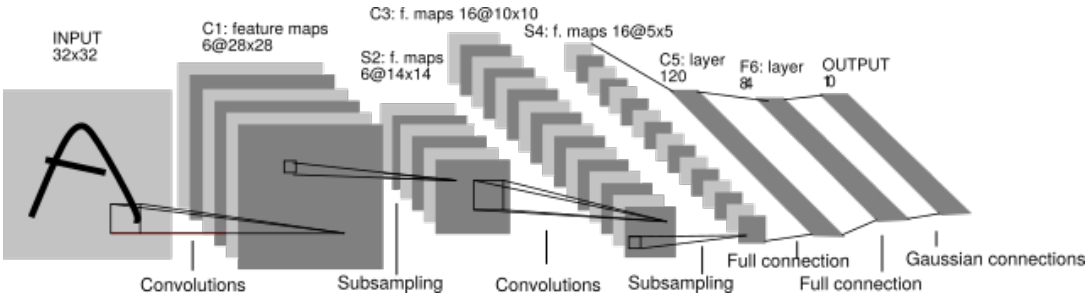


Figure 3.14: Architecture of LeNet-5, which is used for digit recognition (figure taken from [74])

Another very well-known convolutional architecture is the one proposed by Alex Krizhevsky [67], hence usually referred as AlexNet, which improved the state of the art by more than 10% for recognition on ImageNet dataset. The architecture is composed of five convolutional and three fully-connected layers and rectified linear activation is applied to the output of all layers. Max-pooling is used as the subsampling method and drop-out and data augmentation is used to avoid overfitting. The architecture is illustrated in Fig. 3.15 and Fig. 3.16 represents the filters learned in the first convolutional layer of the architecture. Notice that some learned kernels are frequency and orientation selective whereas the others are simply color blobs, due to the restricted connectivity between two parallel

branches in Fig. 3.15, which are processed by two distinct GPUs.

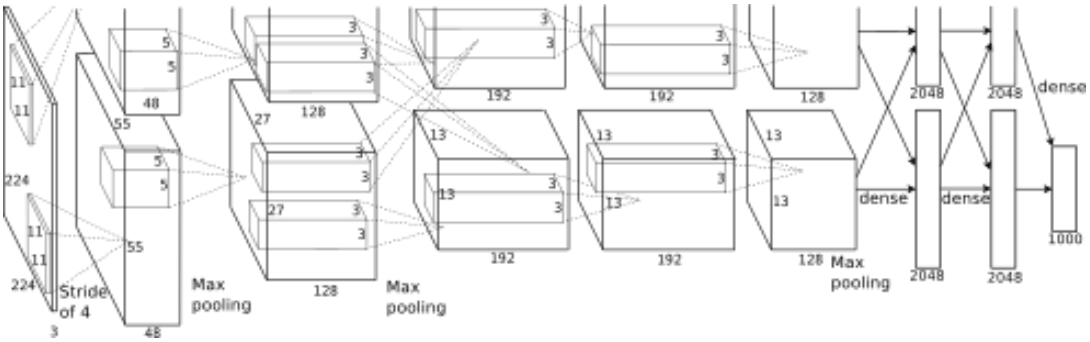


Figure 3.15: Architecture of AlexNet, proposed for Imagenet large scale recognition challange in 2012(figure taken from [67])

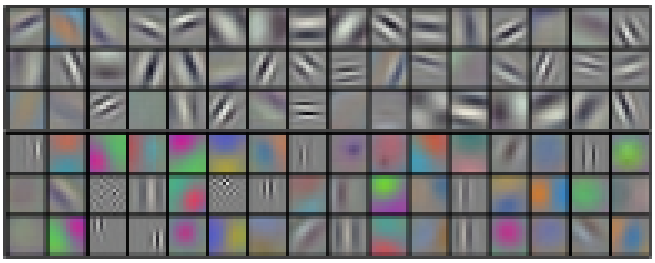


Figure 3.16: Kernels of size 11x11x3 learned in the first convolutional layer of AlexNet [67])

Success of AlexNet has motivated many researchers for deep convolutional architectures; therefore, many variations are proposed especially since 2012. Most of these variations attempt to deepen the architecture, such as GoogLeNet [109] with 22 layers that are a combination of convolutional and pooling layers (both max and average operations are utilized for different layers) that also incorporates dropout and multiscale processing. Although deeper, GoogLeNet is said to use 12 times fewer parameters compared to AlexNet and it took the first place in ILSVRC 2014 by *going deeper with convolutions*. The runner-up in 2014 is also a very-deep architecture, known as VGGNet, which actually accommodates several variations under its name. The variations of VGG are composed of different number of similar layers so that they are used to show that depth is a very critical component. The winner variation contains a total number of 16 layers and convolution and subsampling operations are realized using the same size of kernels throughout the network, 3x3 for convolution and 2x2 for pooling. This winner VGGNet is reported to have nearly 140 million parameters

to be learned, which reveals the importance of optimization involved in training process again. Another relatively different variation [106] is based on the wish for the simplicity, thus, it incorporates only convolutional layers and reLu activations by replacing max-pooling layers with convolutional layers of higher stride values. This work also introduced "guided backpropagation" in order to visualize the features learned not only in the first layer but also in any layer.

3.5 Autoencoders

So far, the structure and learning methods that depend on labeled data are explained. However, as aforementioned before, the amount of unlabeled data grows extremely fast and unsupervised learning intends to exploit this massive data to take advantage of prior knowledge by learning the underlying explanatory factors of low-level sensory data.

Autoencoders are fully connected feed-forward neural networks and they try to learn useful representations that will be used to generate the data itself. In other words, in the most basic form, an autoencoder attempts to learn a function that maps the data to itself. Although this function corresponds to identity at first glance, putting some constraints on the learning process leads to very interesting and useful representations. For example, probably the oldest feature extraction method, Principle Component Analysis (PCA) [90], projects the data onto a orthogonal basis of d_h dimensions, each of which stands for the greatest variance in the training data. Such a projection may be represented as $h = f(\mathbf{x}) = \mathbf{W}^T\mathbf{x} + b$. Hence, the problem of dimension reduction via PCA can be expressed as an autoencoder with a linear layer and undercompleteness constraint, i.e., dimension of the resultant representation is smaller than the dimension of the data.

By a more formal definition, autoencoders define a specific, closed-form feature-extraction function called as "encoder" part, output of which is mapped back to input data by another parametric function, "decoder" part. The parameters of encoder and decoder are learned simultaneously during learning and the objec-

tive is to minimize the reconstruction error that emerges from mapping process. A representative illustration is given in Fig. 3.17. As can be depicted from the given figure, the neural network is trained to maintain information of input data $\mathbf{x}$ and obtain a representation that also reconstructs the input well by $\tilde{\mathbf{x}}$. The weights $\mathbf{W}$ and $\mathbf{W}^*$ are said to be tied if $\mathbf{W}^* = \mathbf{W}^T$ where $\mathbf{W}^T$ stands for the transpose of $\mathbf{W}$.

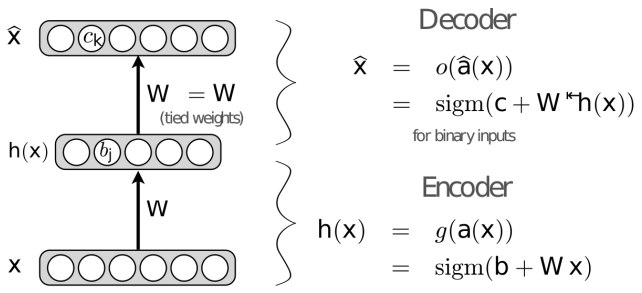


Figure 3.17: Autoencoder illustration: encoder attempts to find an expressive representation $h(\mathbf{x})$ of data $\mathbf{x}$ by minimizing the reconstruction error between the input and reconstructed data $\hat{\mathbf{x}}$.

Two common loss functions used for training auto encoders are cross-entropy and mean square error. Given in Eq. 3.22, cross-entropy is suitable for binary inputs or inputs with values between $(0, 1)$. Cross-entropy is in fact used for measuring the similarity between two probability distributions $p(x)$ and $q(x)$ over the same underlying set of events as $-\sum_x p(x) \log q(x)$. The adaptation is done by simply assuming $p(x = 1) = x_k$ and $q(x = 1) = \hat{x}_k$. As a result, if $x_k = 1$, minimization corresponds to maximizing $\hat{x}_k$ and similarly, $x_k = 0$ means minimizing $\hat{x}_k$. For real valued inputs, mean square error given in Eq. 3.23 is utilized.

$$l(f(\mathbf{x})) = -\sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k)) \tag{3.22}$$

$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2 \tag{3.23}$$

For either case, the gradient of loss function w.r.t pre-activation is $\nabla_{\hat{a}(x^{(t)})} l(f(\mathbf{x}^{(t)})) = \hat{\mathbf{x}}^{(t)} - \mathbf{x}^{(t)}$, and remaining back-propagation process is the same with the one for feed-forward neural networks.

3.5.1 Types of Autoencoders

Linear Autoencoders Linear autoencoders, for which the hidden layer representation is simply a linear function of the input, i.e., $h = f(\mathbf{x}) = \mathbf{W}^T\mathbf{x} + b$, may be trained to obtain either undercomplete or overcomplete representations. Undercomplete representations, which is the case for PCA, result in the "compression" of training data. On the other hand, overcomplete representations can be more problematic for linear autoencoders since each hidden unit may copy a different component of the input and there is no guarantee of meaningful representations. In order to avoid this problem, different constraints are injected to the learning process.

Denoising Autoencoders One interesting approach is to corrupt the input of the autoencoder and attempt to recover the original input from the corrupted version [116]. In other words, the reconstruction $\hat{\mathbf{x}}$ is computed from the noisy input $\tilde{\mathbf{x}}$, however, loss function compares the reconstructed data $\hat{\mathbf{x}}$ with the original input $\mathbf{x}$. The objective is to obtain representations that are robust to noise. The noise process may be chosen to assign random 0 values to input image with probability v, which is known as masking noise, or to add Gaussian noise with standard deviation σ. Structure of denoising autoencoders with masking noise is illustrated in Fig. 3.18.

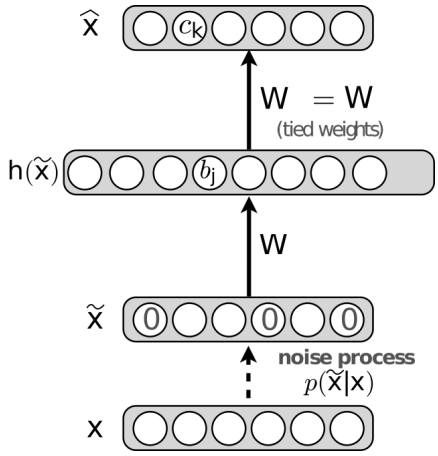


Figure 3.18: In denoising autoencoders, input $\mathbf{x}$ is corrupted with a noise process $p(\tilde{\mathbf{x}}|\mathbf{x})$ and reconstructions is realized over the corrupted input $\tilde{\mathbf{x}}$.

Sparse Autoencoders Another constraint to learn the structure of the input data regardless of the hidden layer size is to introduce sparsity constraint to network. In sparse autoencoders, activity of neurons are forced to be zero most of the times via an additional term in the loss function. The constraint can be chosen as the L_1 norm of the hidden layer activations or Kullback-Leibler (KL) divergence between the mean activation of the network and the desired sparsity. If the hidden layer representation is denoted as $h(\mathbf{x})$, the average activation of unit j over the training set is computed as in Eq. 3.24 . Related KL-divergence term is given in Eq. 3.25 where ρ represents the desired sparsity and generally chosen to be a small number as 0.01 or 0.05. Corresponding L_1 sparsity term is also given in Eq. 3.26.

$$\hat{\rho}_j = \frac{1}{T}\sum_{t=1}^{T}\left[h_j(x^{(t)})\right] \tag{3.24}$$

$$\sum_{j=1}^{j_{max}} \rho \log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j} \tag{3.25}$$

$$\sum_{j=1}^{j_{max}} \frac{1}{T}\sum_{t=1}^{T}|h_j(x^{(t)})| \tag{3.26}$$

Contractive Autoencoders As proposed in [94], the addition of the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input provides a representation that captures the local variations imposed by the data. Proposed additional term attempts to throw away the information of the input data whereas the autoencoder loss function tries to keep that information as much as possible to reconstruct the input data. As a result, the representation obtained by contractive autoencoders is claimed to keep both good and sufficient information. The overall loss function of contractive autoencoders is given Eq. 3.27.

Figure 3.19: Samples from hand-written digits dataset MNIST

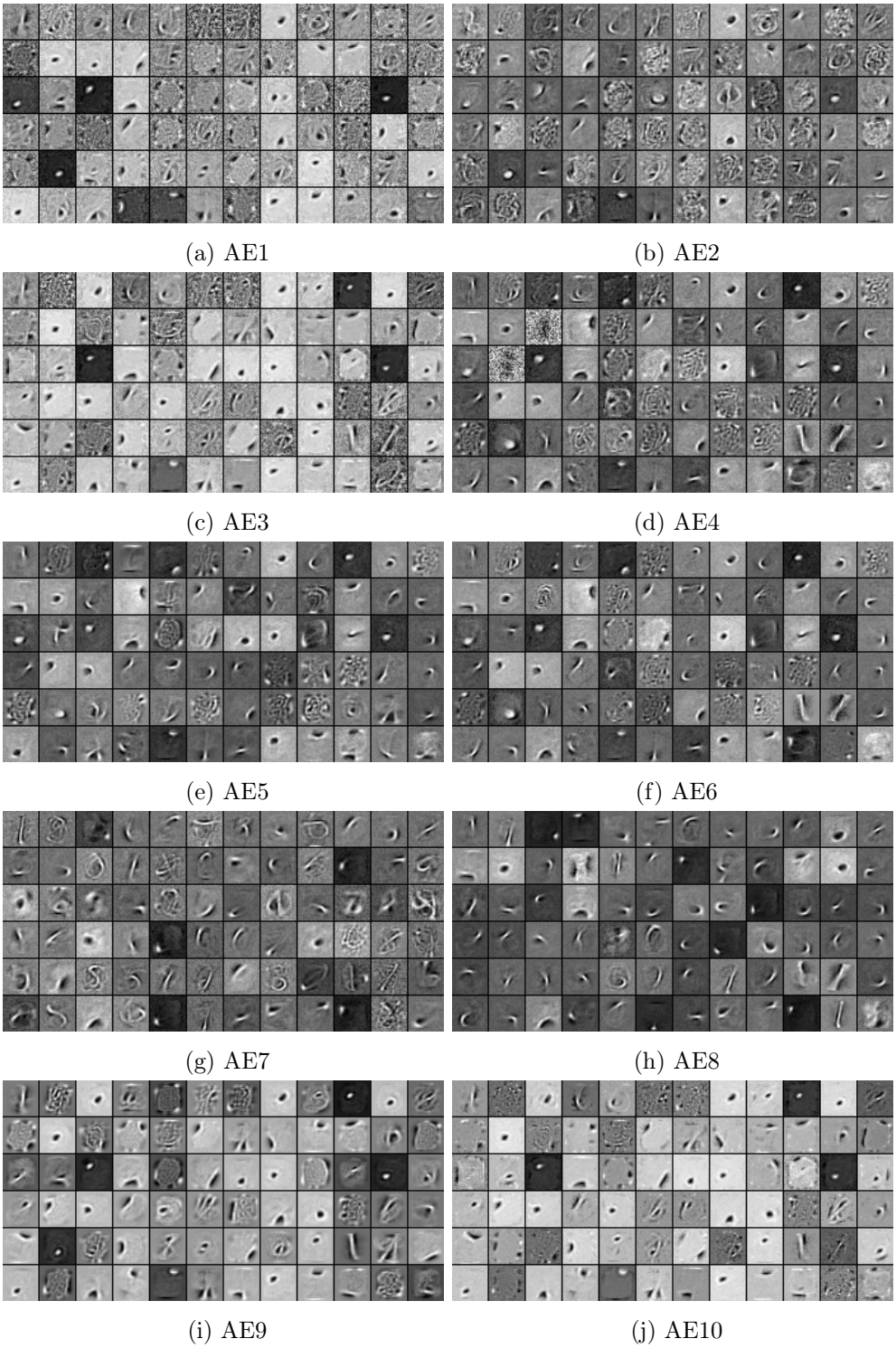
$$l(f(\mathbf{x}^{(t)})) + \lambda \|\nabla_{\mathbf{x}^{(t)}} h(\mathbf{x}^{(t)})\|_F^2 \tag{3.27}$$

$$\text{where} \quad \|\nabla_{\mathbf{x}^{(t)}} h(\mathbf{x}^{(t)})\|_F^2 = \sum_j \sum_k \left(\frac{\partial h(\mathbf{x}^{(t)})_j}{\partial x_k^{(t)}} \right)^2 \tag{3.28}$$

Since autoencoders are feed-forward networks, the same regularization strategies for training feed-forward neural networks, such as weight constraints, also apply for autoencoders. In order to observe the weights learned via different constraints, autoencoders with different loss functions are experimented within a simple framework. The objective is to visualize the filters of an autoencoder with 500 hidden units and the hand-written character dataset MNIST is chosen as the input data. Minimization of several loss functions is achieved via stochastic gradient descent with a constant learning rate (0.25) for a constant number of epochs (25). The general form of the loss function is given in Eq. 3.29 where $l(\mathbf{W})$ stands for the weight regularization term and $l(h(\mathbf{x}))$ is used for sparsity constraint. Configuration of the loss functions with related coefficients is summarized in Table 3.1. The resultant weights are exemplified in Fig. 3.20.

$$\mathcal{L} = \underbrace{l_0(f(\mathbf{x}))}_{\text{data term}} + \lambda_1 \underbrace{l_1(\mathbf{W})}_{\substack{\text{term for} \\ \text{weight regularization}}} + \lambda_2 \underbrace{l_2(h(\mathbf{x}))}_{\text{sparsity term}} \tag{3.29}$$

Whatever the loss function is, the fact that autoencoders attempt to learn the structural information about training data is evident. Visual results may be discussed in terms of their homogeneity or sparsity; however, the performance of the autoencoders is task-dependent, thus hard to interpret. Nevertheless, there are certain things that can be concluded from the given results such as



(a) AE1 (b) AE2

(c) AE3 (d) AE4

(e) AE5 (f) AE6

(g) AE7 (h) AE8

(i) AE9 (j) AE10

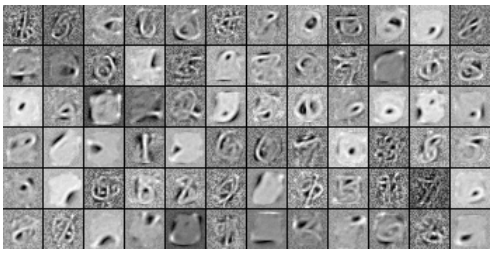
Figure 3.20: Visualization of weights of an autoencoder corresponding to different loss functions which are given in Table 3.1

the effect of regularization. Difference of results between AE2 and AE1, AE4 and AE3, AE12 and AE13 arises from $L2$ regularization, i.e., additional loss

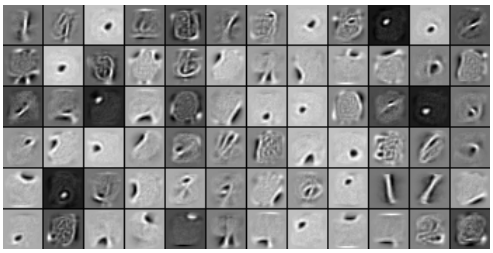
Table3.1: Loss functions of different autoencoder configurations

	Data Term $l(f(\mathbf{x}))$	**Noise Process**	**Weight Reg.** $l(\mathbf{W})$	λ_1	**Sparsity Term** $l(h(\mathbf{x}))$	λ_2
AE1	Mean Square Error	Binomial, $v = 0.5$	None	-	None	-
AE2	Mean Square Error	Gaussian, $\sigma = 0.5$	None	-	None	-
AE3	Cross Entropy	Binomial, $v = 0.5$	None	-	None	-
AE4	Cross Entropy	Gaussian, $\sigma = 0.5$	None	-	None	-
AE5	Cross Entropy	Gaussian, $\sigma = 0.5$	L2	$1x10^{-3}$	None	-
AE6	Cross Entropy	Gaussian, $\sigma = 0.5$	L1	$5x10^{-5}$	None	-
AE7	Cross Entropy	Gaussian, $\sigma = 0.5$	None	-	KL, $\rho = 0.01$	$5x10^{-1}$
AE8	Cross Entropy	Gaussian, $\sigma = 0.5$	L2	$1x10^{-3}$	KL, $\rho = 0.01$	$1x10^{-1}$
AE9	Cross Entropy	Binomial, $v = 0.5$	L2	$1x10^{-3}$	None	-
AE10	Cross Entropy	Binomial, $v = 0.5$	L1	$5x10^{-5}$	None	-
AE11	Cross Entropy	Binomial, $v = 0.5$	None	-	KL, $\rho = 0.01$	$5x10^{-1}$
AE12	Cross Entropy	Binomial, $v = 0.5$	L2	$1x10^{-3}$	KL, $\rho = 0.01$	$1x10^{-1}$
AE13	Cross Entropy	None	None	-	KL, $\rho = 0.01$	2
AE14	Cross Entropy	None	L2	$1x10^{-4}$	KL, $\rho = 0.01$	1

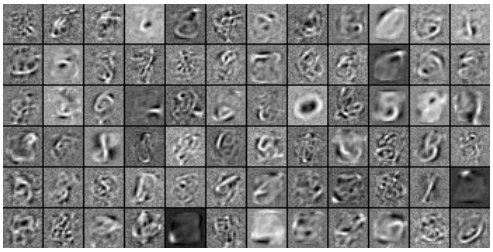
term that forces weights to have small values. This regularization provides some noisy blob like-features to evolve into stroke-like features. Furthermore, addition of sparsity terms provides localization of features as the way AE8 differs from AE7; however, without weight regularization, sparsity itself may go for less to learn structural properties as in AE13. Some samples from MNIST dataset are depicted in Fig. 3.19 for the sake of completeness.



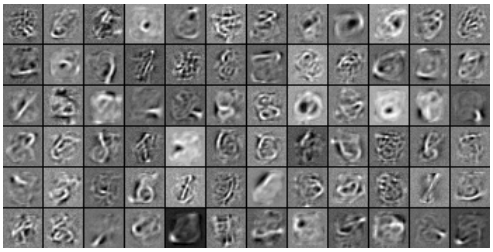
(a) AE11



(b) AE12



(c) AE13



(d) AE14

Figure 3.21: (continued) Visualization of weights of an autoencoder (AE) corresponding to different loss functions which are given in Table 3.1

Denoising autoencoders can also be interpreted within the manifold learning

perspective. N dimensional data usually constitute a lower dimensional topology due to their internal structure. Suppose that the one-dimensional manifold in Fig. 3.22 can represent the space spanned by hand-written digits "2". In other words, all samples of digit "2" are placed near to the manifold in the related space, as represented with crosses in the figure and random samples are further away from the manifold as the noise image in the figure illustrates. The noise process in denoising autoencoders then pushes the samples away from the manifold by the introduction of masking or Gaussian noise. The model is therefore forced to learn the structure so that it can relate the corrupted data to the uncorrupted version, which is possibly *the closest* projection of it onto the underlying manifold.

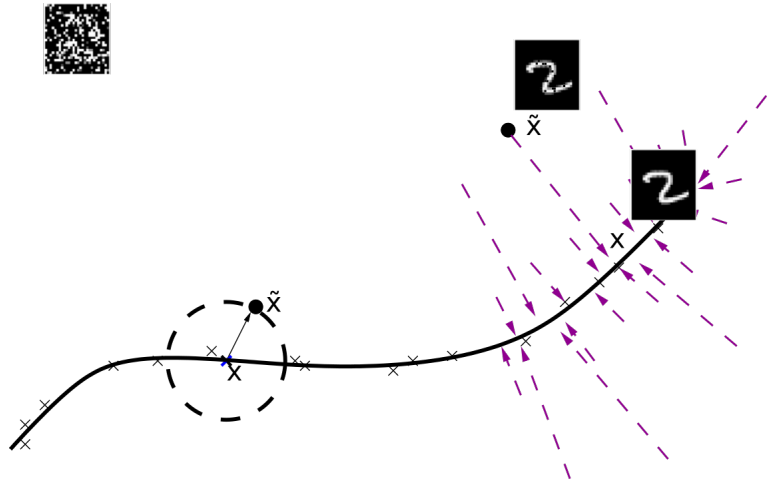


Figure 3.22: Manifold interpretation of denoising autoencoders: Noise process moves the training samples farther away from the manifold so that model is forced to learn the structure (image taken from [69])

CHAPTER 4

DEEP ARCHITECTURES FOR TRACKING

As discussed in detail in Chapter 3 most of the recent approaches in object recognition and detection literature involve deep structures, which have shown superior performance in image classification challenges [44, 67]. Hierarchical structure provides high capacity to learn complex models with multiple levels of abstraction. The robust expressivity obtained by the use of deep structures is used for object detection purpose in different ways. In [110], the geometric information necessary for object detection is captured using a deep neural network-based regression model with a multi-scale box inference. The algorithm is claimed to outperform the state-of-the-art methods for some object classes in Pascal Visual Object Challenge (VOC) 2007 [29] dataset, regarding the average precision criteria used to evaluate the performance of detection algorithms. This algorithm is further improved by training a single deep neural network [28] which jointly trains representations and the bounding box predictors. Resultant class-agnostic method not only achieves competitive results on Pascal VOC 2007 benchmark but also generalizes over unseen classes. Girshick et al. [37] show that a Convolutional Neural Network (CNN) outperforms systems based on HOG-like features in terms of the object detection performance on PASCAL VOC dataset. The localization problem of CNN is attacked by region proposal approach: their method generates several region proposals with corresponding features obtained using CNN, and classification of proposed regions as objects or non-objects is achieved using linear SVMs. The same localization problem of deep Convolutional Neural Network is also attacked via the use of a fully-connected Conditional Random Field [20]. Using the whole algorithm, a more

dense score evaluation is achieved and the resultant label assignment probability is used as a unary term in fully-connected CRF. Results obtained on PASCAL VOC 2012 segmentation benchmark shows that the proposed method is capable of generating semantically accurate predictions. Instead of neural networks, a mixture of hierarchical tree models is used to represent objects in [131]. Detection results on PASCAL VOC 2007 dataset reveal that deep structures are better than shallow structures for object detection.

Despite the huge interest in deep architectures for object recognition, speech processing or scene understanding, the use of deep architectures on object tracking is very limited. The proposals up to now, all of which take different approaches to the problem, will be briefly explained before the explanation of experimental setup and sub-blocks.

4.1 Related Work: Tracking Frameworks that Involve Deep Architectures

This part addresses understanding the proposed visual tracking algorithms that involve deep architectures or related learning methods.

In [59], a two-layer convolutional neural network with a radial basis function at the top layer is used for object detection for tracking. The output vector of the image patch that is annotated in the first frame is used as a positive example. Then, each following frame is divided into small patches, for which output vectors are computed via the neural network. Using the distance between the positive example and patches in a frame, a confidence map is generated. Detection is achieved by selecting the rectangle that corresponds to the peak value in confidence map, which must exceed a pre-defined threshold. Three experiments are conducted regarding the parameter initialization: for the first one, parameters are initialized randomly. For the second, parameters are obtained by a supervised training method by the help of an auxiliary dataset. The third experiment is based on parameters trained in an unsupervised manner, using K-means clustering learning, on another auxiliary dataset. Their experiments

show that the unsupervised training of parameters is better for the framework they proposed.

Convolutional neural networks are also used for human tracking in [57], which is actually out of scope due to the specific object model. However, it is a good idea to understand the way deep architectures are adapted to tracking problem. After offline training of parameters using their task-dependent dataset, a fixed-size rectangle is extracted around the human head annotated in the initial frame. A rectangle at the same position is extracted in the next frame and these two rectangles are fed to a neural network in order to capture the temporal structure. R,G,B channels together with horizontal and vertical gradients are used as feature maps of the rectangles. The proposed algorithm uses two pathways for CNN, one of which tries to capture local structures with a single convolution and a sampling layer while the other pathway intends to reveal global information with consecutive convolution and sampling layers. Four-times upsampling used on the global branch is claimed to bring shift-variant properties. Processing of feature maps that carry information from each patch individually and together with temporal neighbor patches jointly results in a probability map, maximum of which is used as the detection result. In order to adapt scale changes, scale of the bounding box is also estimated using an additional CNN. This object-specific tracker is shown to be more stable in case of drastic view/pose change and false-positive matches.

Another approach exploits an auxiliary dataset for transfer learning. Wang et al. propose the use of stacked denoising autoencoders that are trained offline using natural images [118]. The use of auxiliary data for knowledge transfer is also used in [120], in which an overcomplete dictionary is learned from natural images and used for sparse representation during online tracking. Similarly, Wang et al. use 1 million gray-scale images to train a network of denoising autoencoders that are used to recover from corrupted data. Training data is corrupted with masking occlusion, Gaussian noise or salt and pepper noise and autoencoders are trained using the corrupted images so that the reconstruction error is minimized. The use of denoising autoencoders is claimed to contribute to the robustness of tracker in case of occlusions. During online tracking, a sigmoid classifier is added

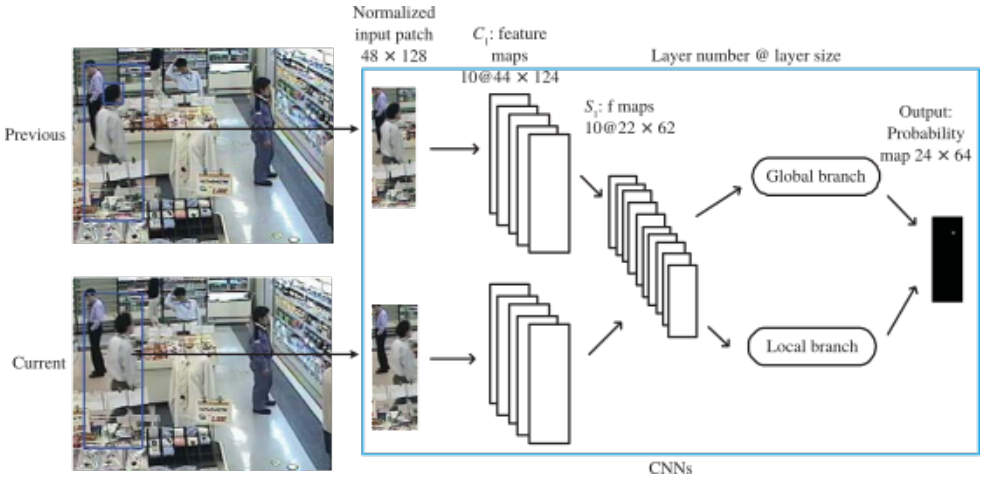


Figure 4.1: Architecture of the CNN utilized in human tracking algorithm [57]. Image patches corresponding to the same position in two consecutive frames are fed to the network which processes the input data in global and local branches. Output of the architecture is a probability map rather than a classification label.

to the autoencoder network and tracking is achieved using particle filters. The experimental results show that the proposed method achieves good success rates with very low center location errors in general. After training with a patch larger than the target, detection of the new position is made on the point that gives the maximum correlation value. Update of the model is improved with a memory using linear interpolation.

Opposite to the findings of [59] and philosophy of deep learning for generalization, a very recent work [75] proposes an online learning scheme for convolutional neural networks. Authors state the difficulty of adopting CNN to visual object problem mainly due to the lack of reliable training samples and overfitting of CNNs to most recent observations. The employment of CNN architecture to proposed tracking by detection scheme is achieved via a special loss function that consists of a truncated norm and a structural term. They also propose a sampling mechanism that takes the label noise problem into consideration and makes use of the frame index for an iterative stochastic gradient descent with a temporal sampling mechanism. The network architecture is composed of two convolutional layers and two fully-connected layers. Two locally normalized image patches for gray images and H and S channels for color images are fed to the network together with the gradient image. The results of the proposed al-

gorithm both on CVPR 2013 benchmark and VOT 2013 challenge demonstrate the best performance for most of the conditions, which indicates the possible good adoptions of deep learning algorithms to visual object tracking problem. The flow chart of the proposed algorithm is illustrated in Fig. 4.2.

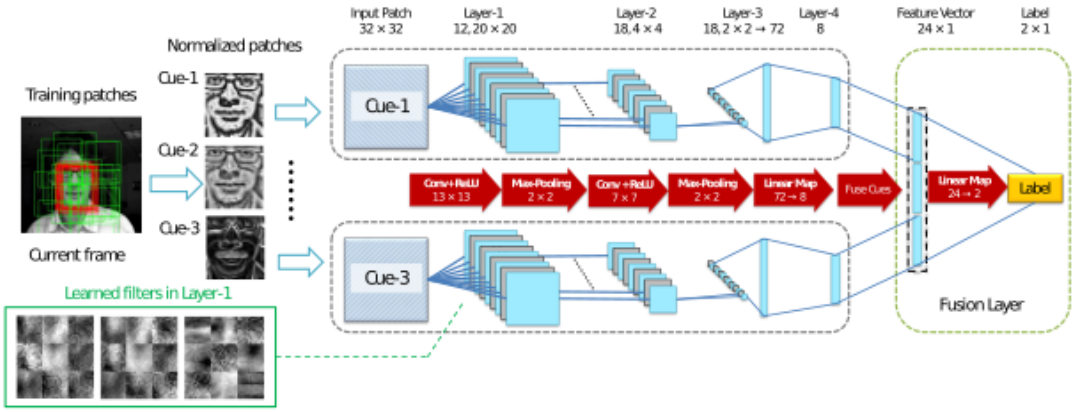


Figure 4.2: Architecture of the CNN utilized in DeepTrack algorithm [75]. Three different cues are fed to the network and independent filters learned through these three different paths. Resultant representations are fused in the top layer where a 2 dimensional feature is generated to indicate class probabilities of being positive or negative.

A totally different approach, which is inspired from the human perceptual system and utilizes an attentional model for tracking, is proposed in [9]. The model is composed of ventral and dorsal pathways. Ventral pathway, known as *what* module, is modeled using (factored) Restricted Boltzmann Machines (RBM) and a particle filter is used for the dorsal pathway which is known as *where* module. The particle filter is used to estimate the location, orientation, speed and scale of the object on the lowest level of dorsal attentional mechanism, and, an online hedging algorithm provides the policy for where to search in the next frame. For the appearance model, a (factored)-restricted Boltzmann machine in the first hidden layer with a hidden layer of multi-fixation RBM on the top, is used on the foveated observations for classification. Their results on both synthetic and natural sequences show that the proposed algorithm is capable of learning where to look from the uniformly distributed gazes in the beginning. The algorithm is improved in [25] with a different gaze control strategy for a better performance in the presence of partial information.

4.2 Tracking by Detection

Tracking by detection is a very common framework for both single and multi object tracking. As for all tracking algorithms, object is defined by its location and the aim is to determine the location in the next frame. Using the object location in the previous frame, motion module, which may be Kalman or particle filter as explained in Chapter 2.3.1, outputs a possible object location regarding the motion dynamics of the system. Generated position hypotheses are converted to candidate object regions according to the appearance model. The decision is then made by comparing the relevance of candidate regions to the object model. This classification result may be utilized to update the appearance model in order to adapt the appearance and environment changes.

In this thesis, we will investigate different object representations for tracking by detection scheme. The framework for experiments is constructed to take the initial frame and corresponding bounding box and obtain object representations to train a classifier. For the initial frame, motion model generates candidate noisy positions around the given location. Once the initial model is constructed, successive frames are processed by considering the patches generated according to the motion model output of the previous frame. Patches are converted to object representations by related algorithm and these representations are fed to a classifier that outputs a confidence value about how likely the given representation is to be the object it learned. For adaptation, an update mechanism is utilized for classifier by generating positive and negative samples according to the previous detection results, which is also possible for adaptive representations. Object detection is achieved using a weighted average of candidate regions with corresponding confidence values. Overall framework with object representation, classification and a motion estimation modules, is illustrated in Fig. 4.3.

4.2.1 Tracking Scheme

Object tracking is achieved within a Bayesian inference framework by describing the measurement model according to the object detection scheme. In general,

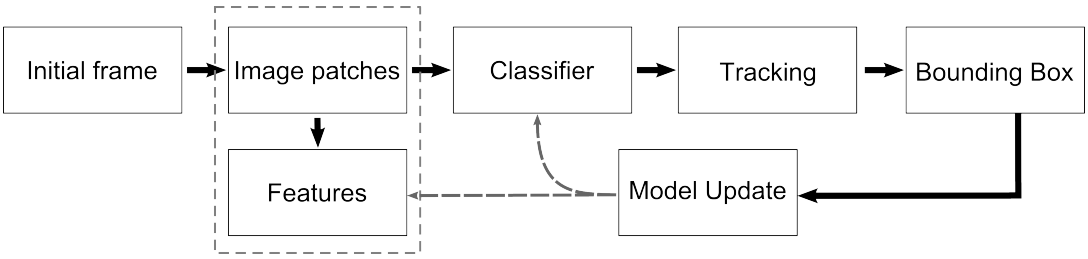


Figure 4.3: Generic framework for tracking by detection algorithms. Solid lines indicate the certain relations between sub-blocks whereas dashed lines are used for optional interaction.

a system model that describes the evolution of *state vector* accompanied by a measurement model for noisy acquirements is required for the analysis and inference of a dynamic system. Although it is possible to construct a posterior probability density function (pdf) based on all the information available, which may be regarded as batch-processing, sequential processing of the received data by recursive filters fits more to visual object tracking problem. Processing may be achieved in two stages: after *predicting* the state pdf from one measurement to the next, current measurement is used to *update* the predicted distribution, which is achieved using Bayes theorem.

In mathematical notation, tracking can be formulated as in Eq. 4.1 if the state sequence up to time t is represented as $\mathbf{x}_{1:t}$ and related measurements are depicted as $\mathbf{z}_{1:t}$, the objective is to find the current target state by maximum a posteriori estimation.

$$\hat{\mathbf{x}}_t = \arg\max_{\mathbf{x}_t} p(\mathbf{x}_t|\mathbf{z}_{1:t}) \tag{4.1}$$

In a recursive filter, we assume that $p(\mathbf{x}_{1:t-1}|\mathbf{z}_{1:t-1})$ at time $t-1$ is available. Thus the distribution we want to estimate can be written as in 4.2, which can then be converted into the expression in 4.3 with Markov assumption, i.e., $p(\mathbf{x}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1})$.

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{z}_{1:t-1}) p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})\, d\mathbf{x}_{t-1} \tag{4.2}$$

$$= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})\, d\mathbf{x}_{t-1} \tag{4.3}$$

The expression in Eq. 4.3 is known as the prediction stage where the probabilistic evolution of state vector, $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, is known as the system model. The prediction stage in the recursive filter may be realized using the relation in Eq. 4.4 once the measurement $\mathbf{z}_t$ is obtained.

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{\int p(\mathbf{z}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{z}_{1:t-1})\, d\mathbf{x}_t} \tag{4.4}$$

The analytic solution of this recursive propagation density is not possible [4]; however, both optimal and suboptimal methods for some cases are present. If the posterior density at each time step is assumed to be Gaussian, Kalman filter is the optimal solution. However, for most of the cases, the strong assumptions on noise and linear system and measurement processes do not hold, which reveals the need for approximations. Recursive Bayesian filter by Monte Carlo simulations, which is known with different names [4] and will be referred here as particle filters, is such an approximation that represents the required posterior density by a set of random samples $\{\mathbf{x}_t^i, i = 1, ..., N_s\}$ and associated weights, $\{\omega_t^i, i = 1, ..., N_s\}$, where N_s demonstrates the number of random samples, i.e., particles. Weights are normalized such that $\sum_i \omega_t^i = 1$ and the discrete weighted approximation of the posterior probability is given in Eq. 4.5.

$$p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) \approx \sum_{i=1}^{N_s} \omega_t^i \delta(\mathbf{x}_{0:t} - \mathbf{x}_{0:t}^i) \tag{4.5}$$

Given that samples are drawn from an importance density $q(\mathbf{x}_{0:t}|\mathbf{z}_{1:t})$, weights are related to the importance density as given in Eq. 4.6. If importance density is chosen to factorize such that $q(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = q(\mathbf{x}_t|\mathbf{x}_{0:t-1}, \mathbf{z}_{1:t}) q(\mathbf{x}_{0:t-1}|\mathbf{z}_{1:t-1})$, sequential update of weights are possible as given in Eq. 4.7, with an assumption of first-order Markov process, i.e., $q(\mathbf{x}_t|\mathbf{x}_{0:t-1}, \mathbf{z}_{1:t}) = q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{z}_t)$ [4].

$$\omega_t^i \propto \frac{p(\mathbf{x}_{0:t}^i|\mathbf{z}_{1:t})}{q(\mathbf{x}_{0:t}^i|\mathbf{z}_{1:t})} \tag{4.6}$$

$$\omega_t^i \propto \omega_{t-1}^i \frac{p(\mathbf{z}_t|\mathbf{x}_t^i)p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)}{q(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i, \mathbf{z}_t)} \tag{4.7}$$

In order to avoid the common problem of weight degeneracy in sequential importance sampling, which indicates all but one particle will have negligible weight after a few iterations, a resampling method is required. With resampling, the information on weights of the samples are conserved by replacing the samples with higher weights with more samples with equal weights and removing the particles with negligible weights, which is depicted in Fig. 4.4. Inverse transform resampling, also known as Smirnov transform [86], is employed as the resampling method for particle filter. The basic idea is to resample a variable from the uniform distribution and use it to generate particle weights from the cumulative distribution function (CDF). The overall procedure is given in Algorithm 4.

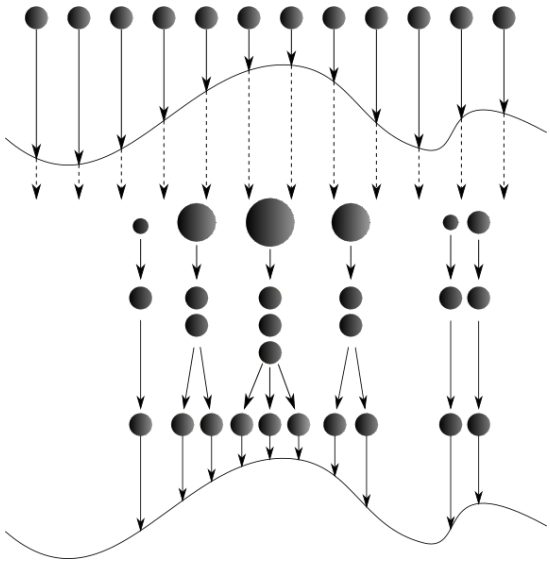


Figure 4.4: Illustration of resampling for particle filter.

The motion model in the framework is characterized as the temporal correlation of the target states in consecutive frames. The state vector represents the affine transformation between consecutive frames with parameters of x and y positions, rotation angle, scale, aspect ratio and skewness. The components of transition density are assumed to be independent and modeled by a zero-mean Gaussian distribution. For the experiments, variance of skewness term is set to zero,

whereas the other variables are kept the same throughout the tracking process. The estimate of the state vector may be achieved in different ways and instead of the commonly used Maximum A Priori (MAP) estimate, which corresponds to the detection of the candidate with maximum confidence value, we employed mean estimate, i.e., $\hat{\mathbf{x}}_t = 1/N_s \sum_{i=1}^{N_s} \omega_t^i \mathbf{x}_t^i$, which clearly introduces smoothness prior to the motion model. The measurement model is determined by the object classification confidence, which will be explained in the next section.

Algorithm 4 Particle Filter

for i from 1 to N_s **do**
  Draw particle $\mathbf{x}_t^i \sim q(\mathbf{x}_t^i | \mathbf{x}_{t-1}^i, \mathbf{z}_t)$
  Assign weight ω_t^i to the particle according to Eq. 4.7
end for
Normalize weights so that $\sum_i \omega_t^i = 1$
Resample particles:
Construct CDF: $c_i = c_{i-1} + \omega_t^i$
Start at the bottom of the CDF: i = 1
Draw a starting sample from uniform distribution: u_1
for j from 1 to N_s **do**
  Move along the CDF: u_j
  while $u_j > c_i$ **do**
    $i = i + 1$
  end while
  $\mathbf{x}_t^{j*} \leftarrow \mathbf{x}_t^i$
  $\omega_t^j \leftarrow 1/N_s$
end for

4.2.2 Classification Scheme

Another important module in the framework is classification, which already has its own literature. The key point for the classification is the fact that our objective is not to achieve strong boundaries between positive and negative samples without regarding the evolution of the boundary. Therefore, after experimenting with support vector machines, which are shown to present better performance

as the top layer of a neural network [111], we decided to utilize a classification method that does not learn from the scratch every time it is trained, rather adapts a decision boundary with new-coming data, such as logistic regression.

Aim of the logistic regression is to learn a function $h_{\mathbf{W}}(\mathbf{x})$, known as sigmoid or logistic, that squashes the value of $\mathbf{W}^T\mathbf{x}$ into the range [0,1] so that the output can be interpreted as probability. Function parameter $\mathbf{W}$ is learned so that the output is high when the data has label "1" and small when input belongs to class "0". Eq. 4.9 summarizes the logistic regression function for input vectors $\mathbf{x}^{(i)}$ and corresponding class labels $y^{(i)} \in \{0, 1\}$.

$$P(y = 1|\mathbf{x}) = h_{\mathbf{W}}(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{W}^T\mathbf{x})} \tag{4.8}$$

$$P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = 1 - h_{\mathbf{W}}(\mathbf{x}) \tag{4.9}$$

Learning of the function parameter $\mathbf{W}$ is achieved via minimizing the cost function given in Eq. 4.10, which measures how well the function $h_{\mathbf{W}}$ predicts class labels. When trained with binary labels $y^{(i)} \in \{0, 1\}$, one of the two terms always becomes zero, i.e., label $y^{(i)} = 0$ corresponds to maximizing $(1 - h_{\mathbf{W}}(\mathbf{x}^{(i)}))$, and thus minimizing $h_{\mathbf{W}}(\mathbf{x}^{(i)}),)$, whereas label $y^{(i)} = 1$ indicates the maximization of $h_{\mathbf{W}}(\mathbf{x}^{(i)})$.

$$J(\mathbf{W}) = -\sum_i \left(y^{(i)} \log(h_{\mathbf{W}}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\mathbf{W}}(\mathbf{x}^{(i)}))\right) \tag{4.10}$$

Notice that the cost function of a logistic regression classifier corresponds to cross-entropy loss for a feed-forward neural network that is trained for binary classification problem. Therefore, choice of the logistic regression is parallel with the aim of adapting neural network representations during training and minimization of the loss is achieved via gradient descent method.

4.2.3 Object Representation

Object representation constitutes the most important part of the scheme since all other modules are frozen during experiments to investigate advantage and disadvantages of various object representation methods. Italic words in bold represents the abbreviations used during performance evaluation.

Raw Pixels (*Intensity*) Intensity values, i.e., gray levels, are used without any inference, only after normalization between [0,1] as a baseline. Results of tracking with intensity values indicate the baseline performance and computational cost of utilized tracking by detection scheme.

Subspace Representation with PCA (*PCA*) Although used as a dimension reduction method in general, principal component analysis is experimented for the purpose of representing images by enhancing the mutual independence of contributory factors. The orthogonal linear transformation of PCA is achieved by using the eigenvectors of the covariance of data and dimension reduction is possible via the elimination of components that corresponds to the amount of data to be discarded. PCA can be regarded as a generative model since the representative eigenvectors, so-called principal components, establish the most of the information in the data; however, they also articulate the greatest variance. Nevertheless, spatial information in the image is not preserved due to the flattening of images to obtain covariance matrix.

PCA is usually utilized for incremental learning algorithms for visual object tracking [98].

Scale Invariant Features (*SIFT*) As another representation, widely-used distinctive image descriptors that are extracted from scale-invariant key points, that are known as SIFT descriptors [79], are utilized. SIFT descriptors are actually three dimensional image gradient histograms that accumulate gradient orientations weighted by gradient norms and a Gauss distribution that indicates the spatial distance from the keypoint. In an extensive study of local image de-

scriptors [85], SIFT features are shown to be distinctive and robust against affine transformations and textured scenes, which makes them reasonable candidates for the tracking problem.

SIFT descriptors have been utilized in visual tracking algorithms both within particle filter framework [30] and keypoint matching methodologies [1].

Pyramid Histogram of Visual Words (*PHOW*) Pyramid Histogram of Visual Words (PHOW) [15] features are similar to SIFT descriptors but extracted at multiple scales, so they may be regarded as an extension of bag-of-words (BoW) that considers the spatial information. For feature extraction, image is divided into sub-regions that becomes finer at the higher levels of so-called pyramids, and dense, quantized SIFT descriptors are computed in each sub-region.

Sparse Representations using Structured Dictionaries (*Sparse Features*) Sparse representations are widely used for visual object tracking as a virtue of their robustness to partial occlusions. For tracking applications, sparse coding methods that aim to model object appearance are claimed to outperform sparse representations that are based on target searching [2]. In this empirical study, we experiment with a reconstructive sparse representation over a structured dictionary, which is generated by applying translation, rotation and anisotropic scaling to a mother function g. Some sample anisotropic refinement atoms [114] that are generated by a mother function $g(x, y) = (2-4x^2)e^{-(x^2+y^2)}$, where x, y indicates the pixel coordinates, are depicted in Fig. 4.5.

Figure 4.5: Examples of anisotropic refinement atoms, which are a subset of the dictionary utilized for sparse representation

Sparse representations over the overcomplete dictionary, which is generated by applying a grid of aforementioned parameters, are obtained using Orthogonal Matching Pursuit (OMP) Algorithm [89]. OMP is a sub-optimal, greedy method

that attacks the well-known sparse representation problem by iteratively selecting the atom that presents the highest correlation with the *residue*. Residue is initialized as the image itself, updated at each iteration and projected onto the span of selected atoms. Iterative search of atoms until the convergence criteria is met. A toy example in Fig 4.6 illustrates the reconstruction of a hand-written digit using different number of atoms obtained by OMP algorithm.

(a) (b) (c) (d) (e) (f) (g)

Figure 4.6: Reconstruction of pattern "3" in (a) by a number of (b) 10, (c) 20, (d) 30, (e) 40, (f) 50 and (g) 100 atoms

For tracking experimentation, coefficients of a pre-defined number of atoms, that results in a sparse vector, is utilized as image representation.

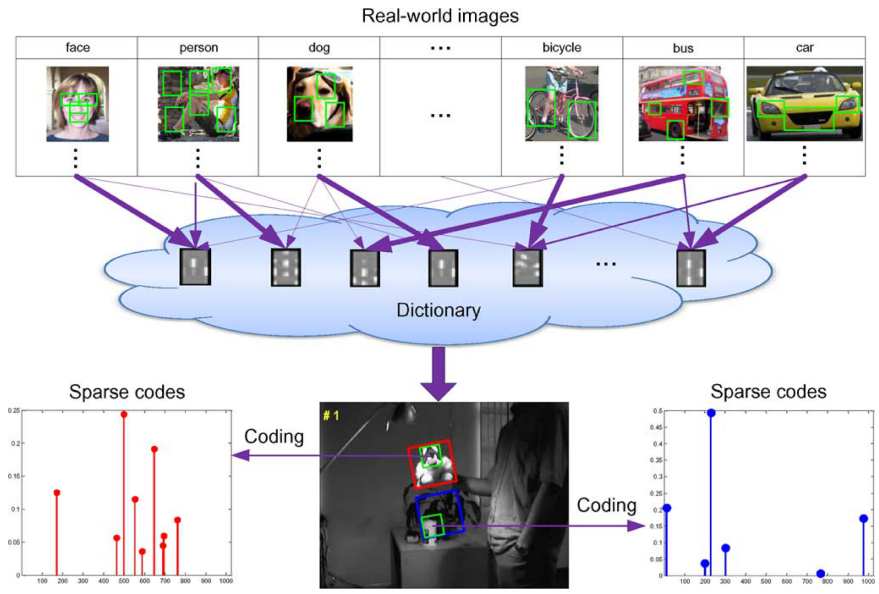


Figure 4.7: Illustration of algorithm in [119], which exploits auxiliary data for learning a dictionary of hand-crafted features.

Transfer Learning with SIFT Dictionaries (*SIFT Sparse*) Parallel with the objective of this thesis, a transfer learning scheme for visual object tracking is presented in [119]. However, learning visual prior from real-world images is different from neural network models and it corresponds to learning a dictionary of sparse codes over SIFT descriptors. For this purpose, SIFT descriptors

from overlapped gray-level patches of object detection datasets are extracted for learning a dictionary is over the descriptors. During tracking, first, SIFT descriptors are extracted from candidate object regions and these features are encoded using the learned dictionary. Final representation is acquired by dividing the candidate patch into non-overlapped regions, max-pooling the coding results of descriptors and concatenating the pooled features from all regions. This highly complex hand-crafted feature based transfer learning method is illustrated in Fig. 4.7.

Stacked Denoising Autoencoders ($\boldsymbol{DAE}$), ($\boldsymbol{DAE}_{gray}$), ($\boldsymbol{DAE}_{color}$), ($\boldsymbol{DAE}_{gray,w/adapt}$) Two different denoising autoencoders with masking noise and L2 regularization are trained using the combination of CIFAR-10 and CIFAR-100 datasets [66], each of which consists of 50000 32x32 color images corresponding to different objects. Figure 4.8 illustrates some random samples from CIFAR-10 dataset.

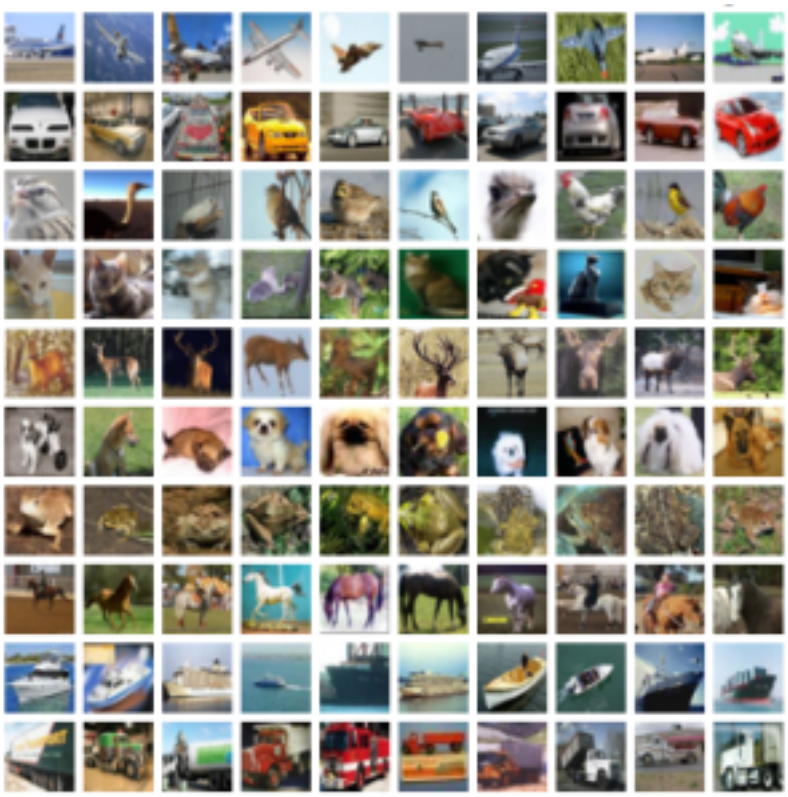


Figure 4.8: Samples from CIFAR-10 dataset [66]

We experimented with three different autoencoders in four different ways. The

first autoencoder is composed of 4 hidden layers and it is a denoising autoencoder which employs additive Gaussian noise process [118] (*DAE*). The second autoencoder has a lower capacity with three hidden layers that consist of less number of neurons and training is achieved using CIFAR-10 and CIFAR-100 intensity images (DAE_{gray}). A similar architecture, same number of layers but larger number of neurons per-layer, is trained using the same dataset but using the color information (DAE_{color}). These three autoencoders are utilized to initialize the corresponding feed-forward network which is fine-tuned in the initial frame and fine-tuned weights are kept the same throughout tracking, except the top layer, which corresponds to logistic regression classifier. The fourth experiment is conducted by adapting not only the last layer but also the weights of lower layers when classifier requires re-training ($\text{DAE}_{gray,w/adapt,}$).

CNN Features off the Shelf *(CNN)* Since the number of training examples in online tracking can be hardly made on the order of hundred, using CNN features off-the-shelf makes more sense instead of training it from scratch or fine-tuning a large-capacity network with limited number of examples that would cause memorization. Therefore, the features obtained before the first fully-connected layer of AlexNet [67], are utilized as CNN features.

4.2.4 Overview of The Framework

For the investigation of different object representations for visual tracking problem, a tracking-by-detection scheme is constructed that is composed of representation, classification and motion estimation modules. The general overflow is as follows:

Initialization

- Initial frame of a sequence is provided with an annotated bounding box, which is illustrated in color in Fig. 4.9a for *woman* sequence. Using the provided bounding box, positive and negative samples are generated for

(a) (b)

Figure 4.9: (a) Initial frame of *woman* sequence in color with annotated bounding box and (b) some of corresponding positive and negative samples, within green and red bounding boxes respectively, in gray level for the training of the classifier.

classifier training. Positive samples correspond to the bounding boxes of the same size as the ground truth which are translated a few pixels with respect to the provided location. On the other hand, negative samples of the same size are extracted around the groundtruth in such a way that they contain small parts of the object. Positive samples for the initial frame in Fig. 4.9a are illustrated with green bounding boxes in Fig. 4.9b where red bounding boxes correspond to some of extracted negative samples.

- All bounding boxes, i.e., the ground-truth, positive samples and negative samples, are converted to intensity image patches of size 32x32, except for CNN features. Intensity values are normalized between [0,1]. Warped samples corresponding to the bounding boxes provided in Fig. 4.9b are illustrated in Fig. 4.10.

- Logistic regression parameter is obtained via gradient descent using this small labeled dataset.

- Any initialization regarding the experimented representation, such as dictionary generation for sparse representation or loading parameters for any pre-trained model, is realized. This step corresponds to the fine-tuning of fully-connected neural networks whose weights are initialized as the

weights of trained autoencoders.

- Particles are generated around the ground-truth using the independent zero-mean Gauss distributions of each state parameter. The variance of skewness parameter is set to zero. Centers and some of corresponding bounding boxes for the first frame particles are illustrated in Fig. 4.11.

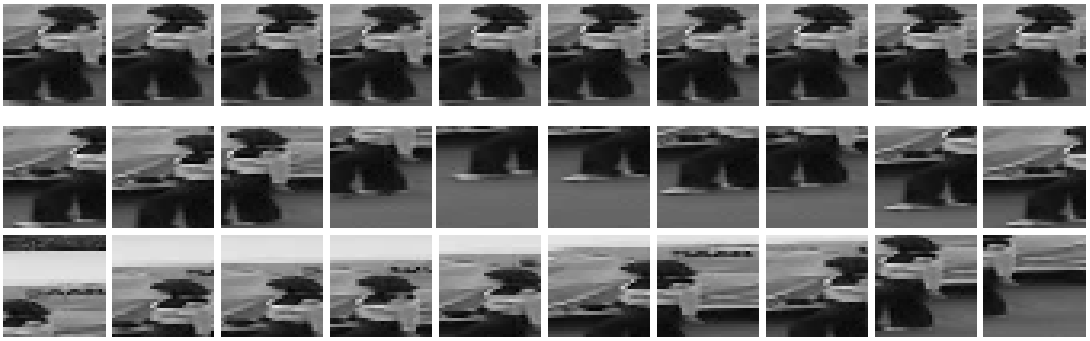
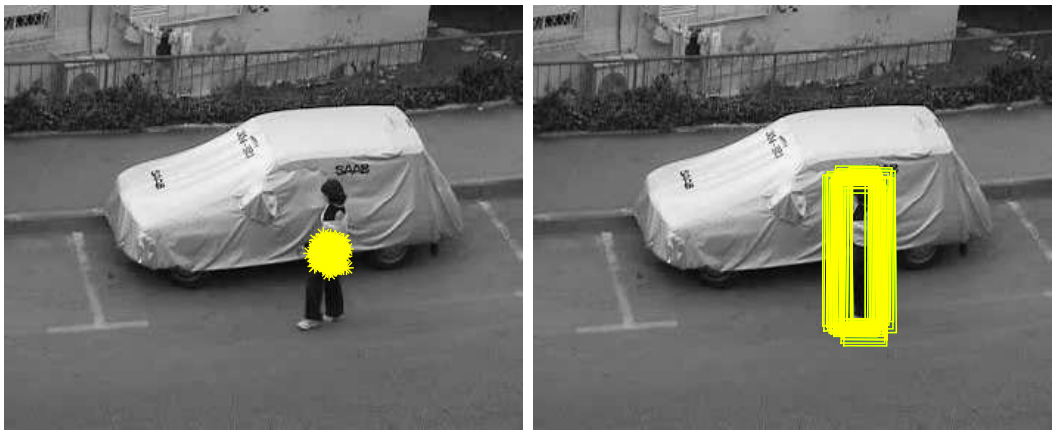


Figure 4.10: Positive and negative samples in the first and latter two rows, respectively, warped into 32x32 gray-level images.



(a) (b)

Figure 4.11: (a) Centers and (b) some of corresponding bounding boxes of first-frame-particles.

Update

- Once the new frame is provided to the algorithm, object candidate rectangles are generated by the prediction stage of the particle filter. Centers and some of corresponding bounding boxes for the particles in 104^{th} frame are illustrated in Fig. 4.12

(a) (b)

Figure 4.12: (a) Centers and (b) some of corresponding bounding boxes of the particles in 104^{th} frame.

- Representation of candidate regions are obtained according to the chosen scheme.
- Representations are fed to classifier and resultant positive-class probability is used to update the particle weights for the next frame.
- Mean estimate of the particles is computed to be the new bounding box of the target object, which is depicted in Fig. 4.13.
- If the highest confidence value is lower than an empirical threshold, which indicates a dramatic change in the view, or the classifier has not been updated for a large number of frames, a classifier update procedure is realized.
- For updating the classifier, positive samples are obtained with memory, i.e., the detection results of the last n_f frames are utilized as positive samples. However, negative samples are obtained from the current frame. Positive and negative samples corresponding to 104^{th} frame in *woman* sequence are given in Fig. 4.14, since it exemplifies the case in which classifier requires an update.
- The current detection result is stored in the positive samples and update procedure is finalized.

Figure 4.13: Detection result of 104^{th} frame in *woman* sequence.

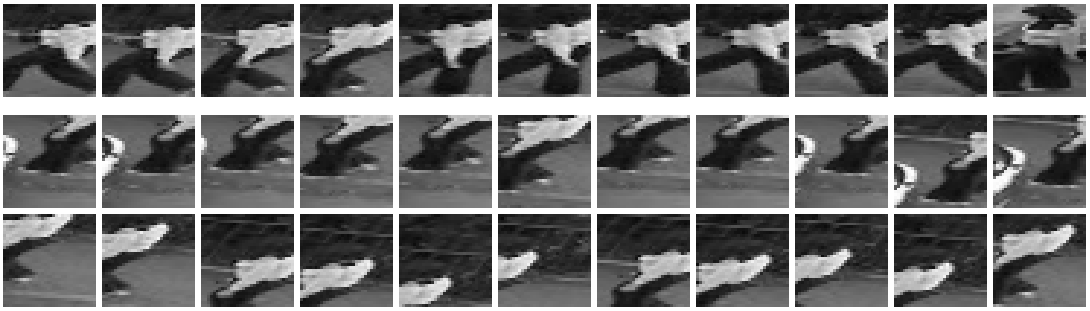


Figure 4.14: Positive and negative samples for 104^{th} frame, in the first and latter two rows, respectively, warped into 32x32 gray-level images. Notice that positive samples correspond to the detection results of the last 10 frames accompanied by the labeled sample in the initial frame.

4.3 Implementation Details

The overall framework is constructed in Matlab with a modular structure. Except from sparse SIFT features [119], which are extracted using the source code provided by the authors, all representations are implemented for the experiments in this thesis.

Training of autoencoders is realized using Theano [6], a Python library which is convenient for deep learning research with many benefits, such as symbolic differentiation and efficient use of computational sources. The structure of the gray-level autoencoder, DAE_{gray}, is chosen to be 1024-896-512-256-1, including input and output layers, whereas color autoencoder, DAE_{color}, utilizes 3072-2048-1024-512-1 structure. Both autoencoders benefit L2 regularization and

optimization is achieved via stochastic gradient descent with momentum of value 0.9. Annealing learning rate strategy is preferred with a starting value of 0.04. A total of 120.000 images are exploited using mini-batch size of 64 for 400 epochs. Binomial noise with corruption probability of 0.4 is employed as the noise process in denoising autoencoders. Fine-tuning is achieved via backpropagation within the Matlab framework.

The number of particles utilized for the estimation of posterior probability in tracking by detection scheme is 1000, except sparse features and sparse SIFT features. Due to the computational burden, 400 particles are utilized for the tracker with sparse SIFT representation whereas only 200 particles are employed during the tracking with sparse features.

4.4 Experimental Results

As stated earlier, all representations are evaluated within the same tracking by classification scheme using the VOT Challenge 2014 toolkit [65]. Toolkit evaluates the algorithms on 25 short sequences that are per-frame annotated with occlusion, illumination change, motion change, size change and camera motion. The ultimate aim of ranking algorithms is achieved via 15 iterations of all sequences and ranking of the algorithms is determined in terms of accuracy and robustness. In this thesis, we utilized the platform in order to run the algorithm once for all sequences so that the toolkit realizes reinitialization when the track is lost. First, some visual results that demonstrate the output bounding boxes of each algorithm will be presented. Then, we will declare the average accuracy and robustness values per sequence, instead of the accuracy and robustness ranks of the algorithms.

4.4.1 Visual Results

This section will include some examples from sequences on which the experiments carried on.

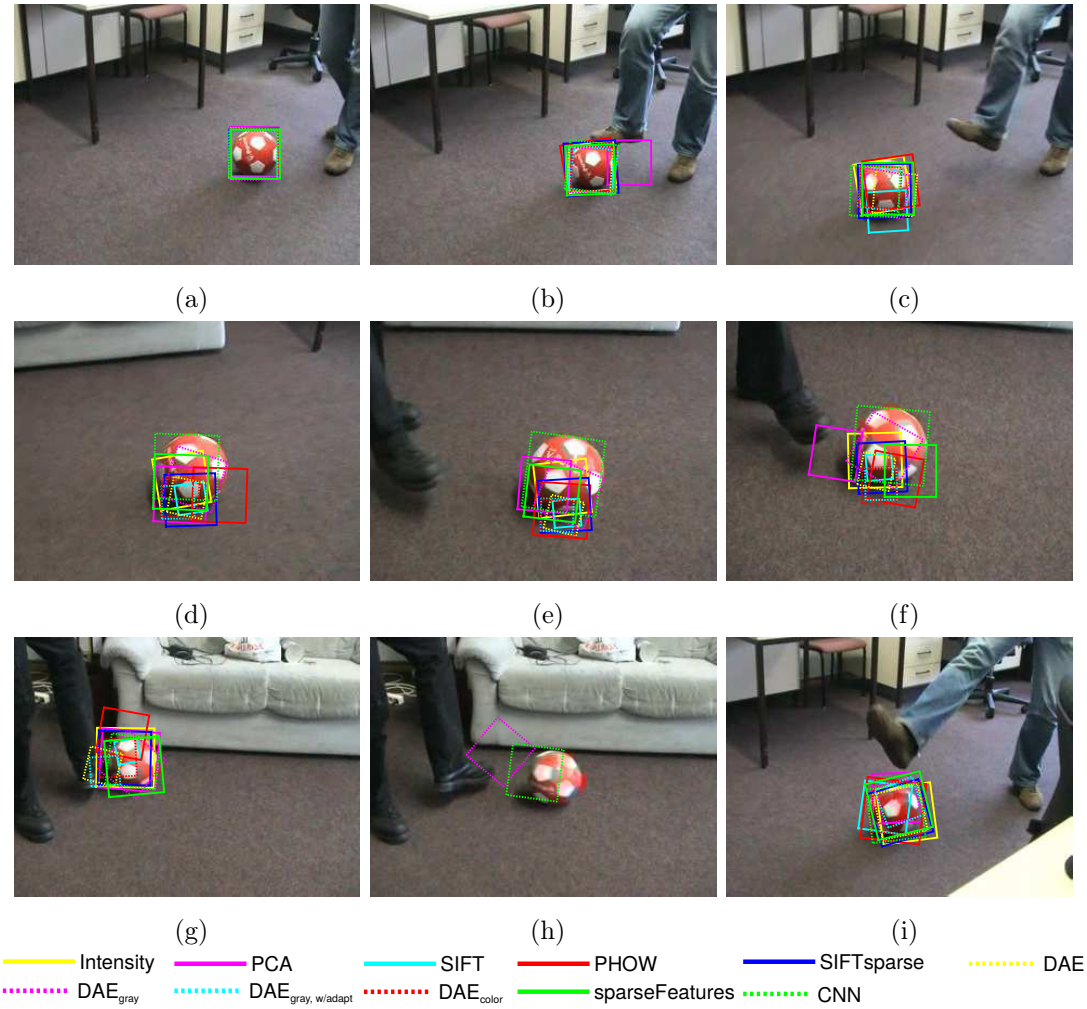


Figure 4.15: Tracking result of sequence ball corresponding frames (a)5 (b)59 (c)78 (d)114 (e)130 (f)146 (g)183 (h)187 (i)249.

ball Ball is a relatively simple sequence that mainly includes motion and scale changes., As depicted in Fig. 4.15, all trackers corresponding to different representations are capable of tracking the target with some accuracy in the very first frames where only a smooth camera motion is present. During the motion of the ball, representations that are based on local descriptors are observed to stuck on local parts of the object, whereas tracker with CNN features capture the global appearance fairly well. However, none of the trackers is able to survive when a fast motion change occurs, and this is probably due to the limitations of motion model.

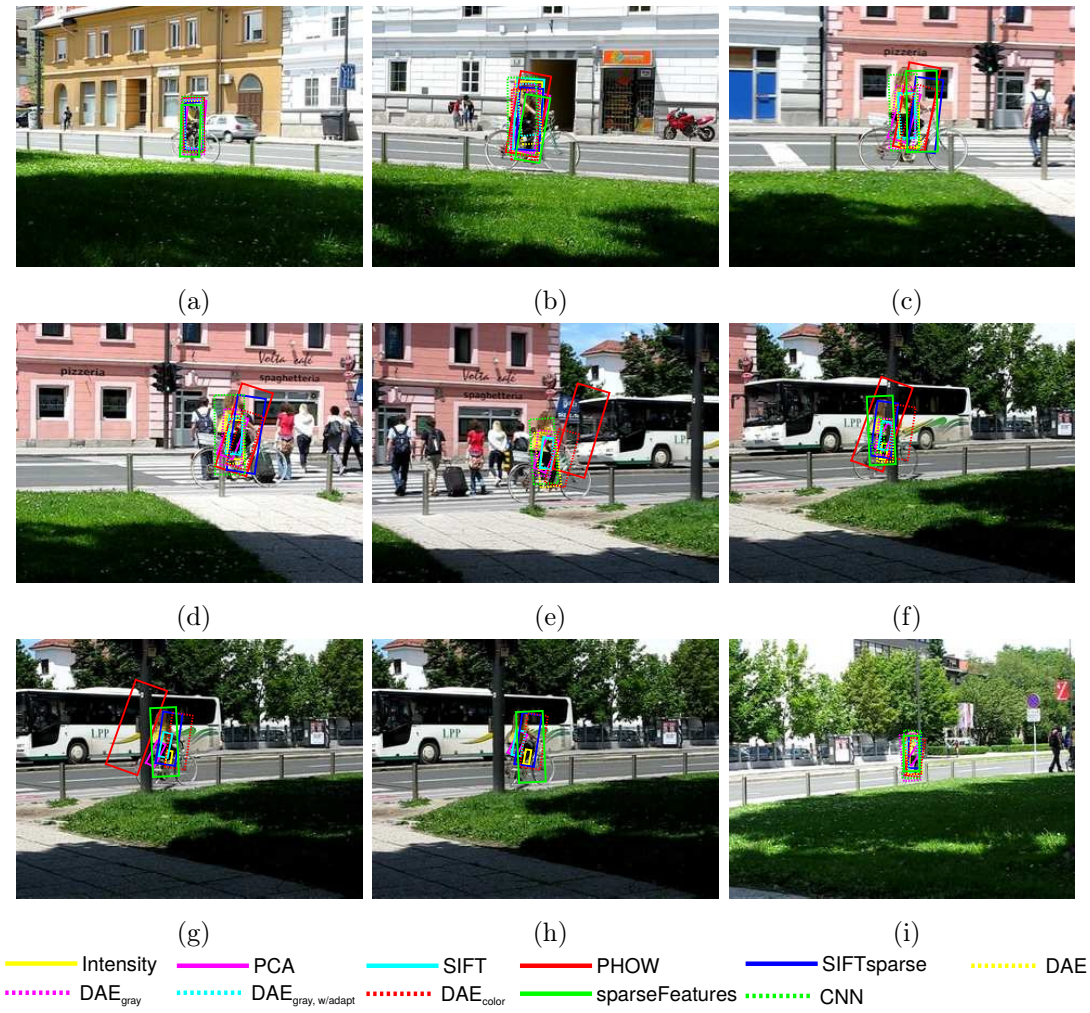


Figure 4.16: Tracking result of sequence *bicycle* corresponding frames (a)10 (b)66 (c)118 (d)135 (e)149 (f)171 (g)176 (h)177 (i)233.

bicycle is a sequence that demonstrates viewpoint and scale change as well as a short-time occlusion. Fig. 4.16 also represents the success of the tracker with CNN features in terms of the size and orientation of the bounding box, until the occlusion that occurs between frames 170-175. Trackers with *sparseFeatures* and colored autoencoder, DAE_{color}, are observed to handle the occlusion better than the trackers with other representations.

david The sequence *david* starts under bad illumination conditions and the object is subject to motion, scale, pose and illumination changes. As Fig. 4.17 indicates, all trackers with local or hierarchical representations are capable of robust visual object tracking under poor illumination conditions, even without

any preprocessing.

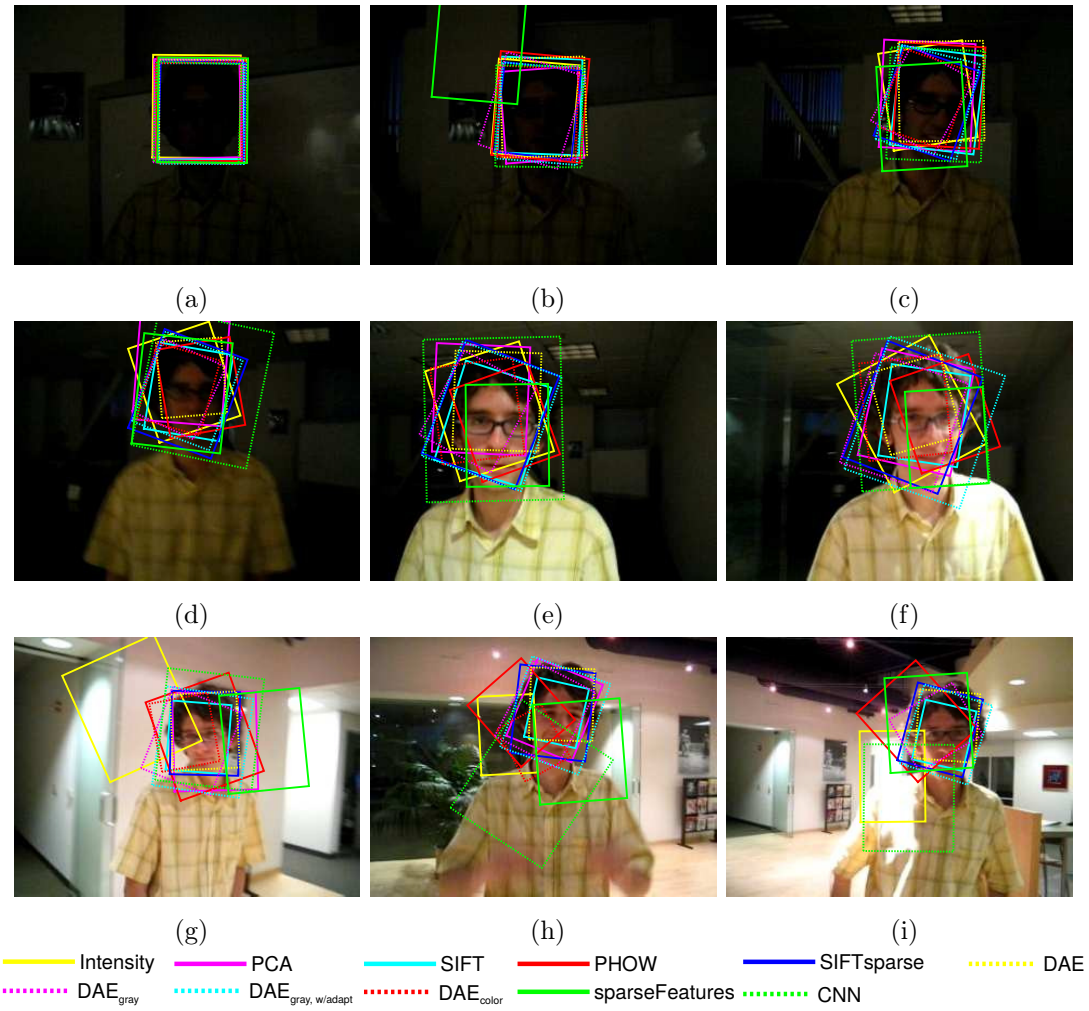


Figure 4.17: Tracking result of sequence *david* corresponding frames (a)10 (b)45 (c)130 (d)200 (e)310 (f)356 (g)426 (h)607 (i)720.

car The sequence *car* demonstrates the scale and pose change as well as an occlusion. Results in Fig. 4.18 present that none of the representations is able to lead the tracker to handle scale and pose change with a bounding box that adapts itself fairly well to cover the most of the object and the least of the background. However, trackers with hierarchical representations demonstrate very good performance in terms of robustness.

gymnastics In the challenging sequence *gymnastics*, a fast motion with rotation resulting in heavy pose changes is present and results are given in Fig.

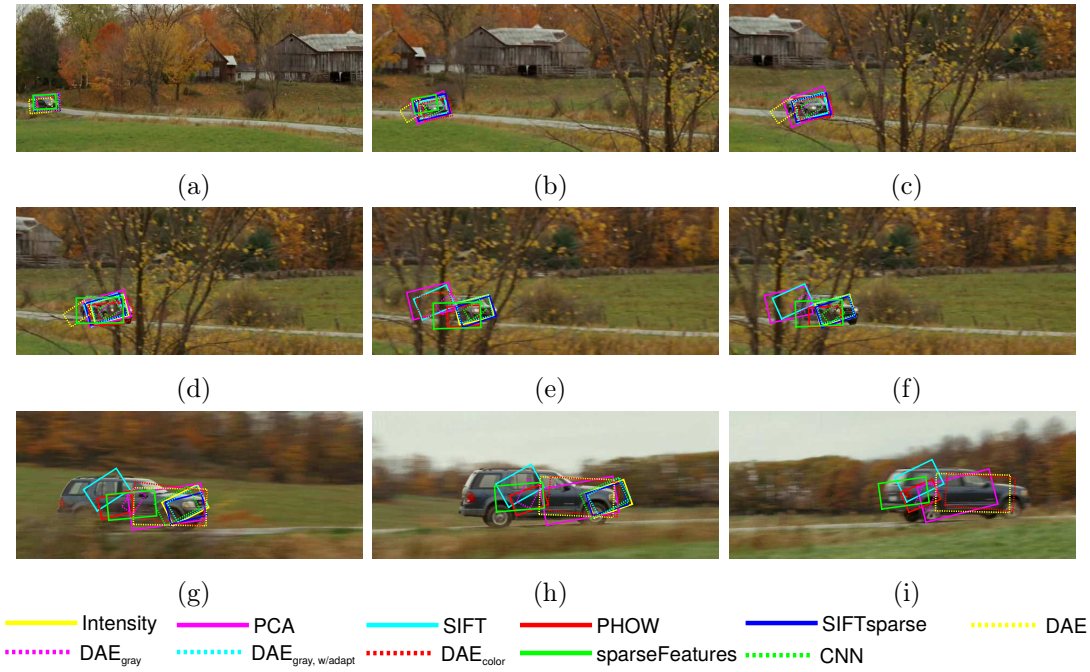


Figure 4.18: Tracking result of sequence *car* corresponding frames (a)22 (b)107 (c)147 (d)159 (e)168 (f)171 (g)218 (h)238 (i)250.

4.19.

4.4.2 Accuracy and Robustness Results

Despite the abundance of performance measures in visual tracking, VOT challenge toolkit utilizes two orthogonal performance measures, accuracy and robustness. Accuracy indicates the amount of overlap between the groundtruth A_t^G and predicted bounding boxes A_t^T as in Eq. 4.11.

$$accuracy = \frac{A_t^G \cap A_t^T}{A_t^G \cup A_t^T} \tag{4.11}$$

Robustness, on the other hand, is determined by the number of times tracker loses the target. Regarding the use of VOT challenge toolkit for the reinitialization of trackers in case of target loss, notice that the accuracy results are biased towards the algorithms that are not robust, due to the re-initialization with ground truth bounding box.

Average accuracy and robustness values of representations are given in Table

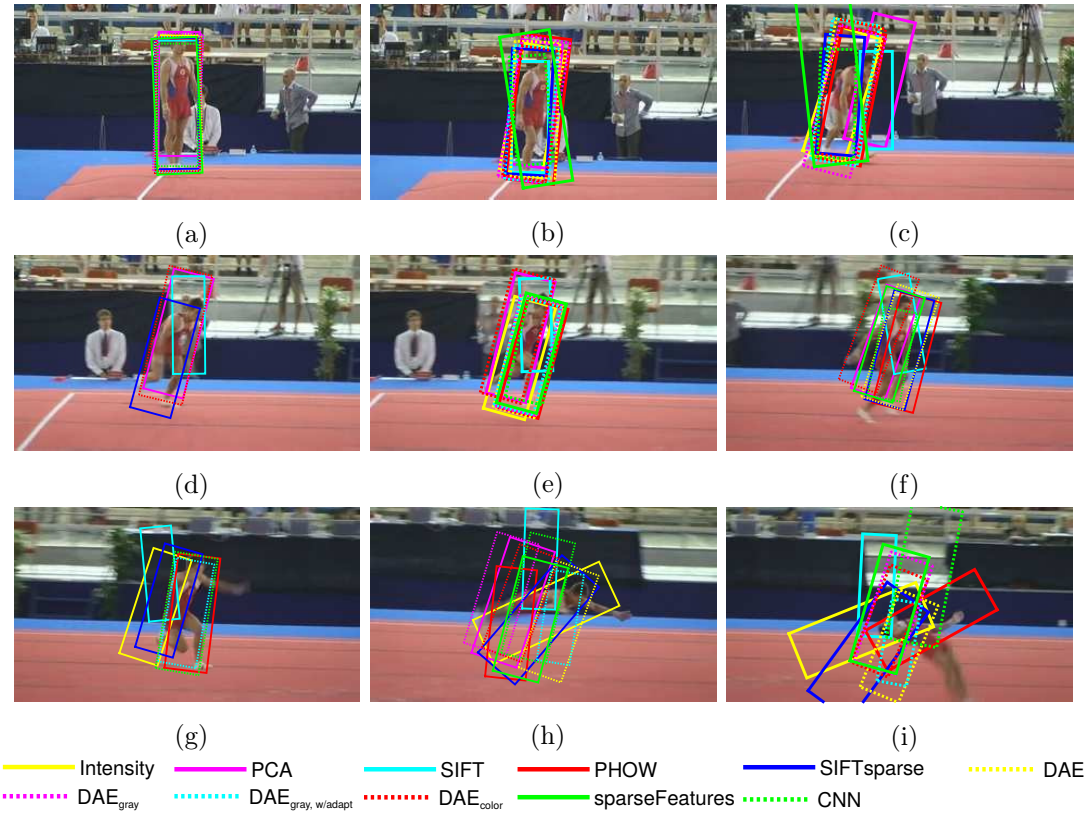


Figure 4.19: Tracking result of sequence *gymnastics* corresponding frames (a)10 (b)74 (c)87 (d)97 (e)103 (f)117 (g)129 (h)140 (i)154.

4.1 and Table 4.2-Table 4.3 respectively. Accuracy is given in terms of average overlap percentage over sequences whereas the first term in robustness values represent the number of failures and the second term indicates the percentage of failures regarding the number of frames in a sequence. For a fair comparison, same values for the four algorithms tested for VOT Challenge 2014, two of which ranked the top two place (DSST [24] and SAMF [77]) in terms of joint ranking and the other two ranked the bottom (MIL [5] and CT [127]), are also provided in the tables.

First of all, experimented tracking scheme is neither as successful as the state-of-the-art methods as DSST and SAMF, nor worse than the last-ranking methods like MIL and CT, which indicates the validity of the experimentation framework. Relatively poor overall performance is also not surprising since the procedure is not tailored to any of the representations and parameters are chosen to be loose in order to better observe the representation power of the algorithms. In other

Table4.1: Average accuracy values of algorithms on sequences

	Intensity	PCA	SIFT	PHOW	Sparse Features	Sparse SIFT	DAE	DAE_{color}	DAE_{gray}	$DAE_{gray,w/adapt}$	CNN	MIL	CT	DSST	SAMF
ball	0.63	0.52	0.36	0.48	0.67	0.56	0.48	0.67	0.67	0.51	0.53	0.43	0.39	0.57	0.78
basketball	0.45	0.31	0.54	0.37	0.35	0.3	0.36	0.4	0.3	0.32	0.37	0.61	0.54	0.64	0.75
bicycle	0.46	0.35	0.47	0.61	0.52	0.57	0.58	0.48	0.58	0.45	0.68	0.54	0.56	0.58	0.62
bolt	0.45	0.44	0.52	0.47	0.37	0.54	0.62	0.54	0.52	0.38	0.38	0.52	0.42	0.56	0.56
car	0.54	0.57	0.5	0.55	0.36	0.54	0.27	0.59	0.31	0.48	0.34	0.42	0.37	0.74	0.51
david	0.47	0.58	0.59	0.51	0.45	0.64	0.64	0.45	0.56	0.71	0.45	0.5	0.4	0.81	0.82
diving	0.28	0.32	0.31	0.33	0.31	0.33	0.29	0.35	0.34	0.33	0.3	0.25	0.24	0.44	0.25
drunk	0.41	0.33	0.32	0.35	0.28	0.34	0.33	0.3	0.4	0.59	0.56	0.49	0.48	0.55	0.57
fernando	0.31	0.36	0.3	0.44	0.27	0.37	0.38	0.36	0.38	0.31	0.43	0.46	0.39	0.34	0.39
fish1	0.45	0.33	0.45	0.36	0.33	0.53	0.47	0.17	0.45	0.34	0.16	0.39	0.36	0.32	0.5
fish2	0.24	0.22	0.3	0.31	0.28	0.34	0.37	0.41	0.23	0.29	0.46	0.2	0.21	0.35	0.3
gymnastics	0.57	0.54	0.47	0.59	0.48	0.53	0.6	0.57	0.56	0.54	0.54	0.28	0.48	0.63	0.54
hand1	0.41	0.36	0.52	0.54	0.18	0.47	0.47	0.5	0.41	0.47	0.43	0.51	0.32	0.21	0.55
hand2	0.27	0.28	0.28	0.36	0.32	0.4	0.27	0.29	0.24	0.28	0.46	0.42	0.2	0.53	0.46
jogging	0.46	0.45	0.55	0.63	0.51	0.75	0.56	0.45	0.58	0.7	0.59	0.2	0.77	0.79	0.82
motocross	0.33	0.46	0.4	0.29	0.28	0.37	0.54	0.33	0.41	0.42	0.41	0.32	0.22	0.42	0.4
polarbear	0.47	0.27	0.55	0.42	0.50	0.52	0.23	0.38	0.3	0.31	0.51	0.46	0.6	0.64	0.71
skating	0.38	0.36	0.48	0.38	0.46	0.53	0.27	0.31	0.27	0.45	0.29	0.29	0.51	0.59	0.45
sphere	0.64	0.56	0.13	0.42	0.16	0.44	0.68	0.38	0.5	0.35	0.46	0.55	0.61	0.93	0.88
sunshade	0.66	0.72	0.64	0.63	0.51	0.69	0.69	0.66	0.51	0.53	0.64	0.42	0.41	0.78	0.76
surfing	0.69	0.6	0.71	0.73	0.66	0.8	0.71	0.8	0.74	0.76	0.71	0.38	0.66	0.91	0.8
torus	0.37	0.4	0.53	0.58	0.28	0.52	0.53	0.59	0.44	0.51	0.66	0.53	0.55	0.81	0.84
trellis	0.58	0.47	0.58	0.43	0.29	0.74	0.53	0.48	0.52	0.64	0.45	0.44	0.33	0.81	0.83
tunnel	0.61	0.47	0.51	0.51	0.48	0.5	0.48	0.56	0.61	0.65	0.56	0.35	0.2	0.81	0.55
woman	0.62	0.59	0.53	0.52	0.54	0.66	0.43	0.61	0.4	0.46	0.62	0.26	0.57	0.79	0.76
average	**0.48**	**0.43**	**0.47**	**0.46**	**0.40**	**0.51**	**0.45**	**0.45**	**0.46**	**0.50**	**0.48**	**0.42**	**0.43**	**0.64**	**0.64**

Table4.2: Number of failures for each algorithm for all sequences

	Intensity	PCA	SIFT	PHOW	Sparse Features	Sparse SIFT	DAE	DAE_{color}	DAE_{gray}	$DAE_{gray,w/adapt}$	CNN	MIL	CT	DSST	SAMF
ball	2	4	2	5	3	3	2	1	2	1	2	1	1	1	1
basketball	3	2	1	5	5	3	2	3	0	2	2	2	1	1	0
bicycle	0	0	1	2	7	2	1	0	1	1	1	0	2	0	0
bolt	3	7	1	3	4	4	4	1	3	2	2	6	9	1	2
car	0	1	0	1	2	1	1	1	0	0	0	0	0	0	0
david	2	1	0	1	3	2	0	1	0	0	0	0	1	0	0
diving	2	3	2	3	3	2	2	1	1	3	2	1	2	1	4
drunk	0	1	1	1	2	0	0	0	0	0	0	0	0	0	0
fernando	1	3	3	3	4	3	4	3	4	1	2	2	3	1	1
fish1	3	8	3	6	10	3	5	1	4	2	1	2	10	1	3
fish2	6	5	4	6	8	3	6	6	7	6	4	6	4	4	5
gymnastics	4	2	2	4	4	4	4	2	5	3	2	5	4	5	2
hand1	4	7	4	4	9	3	4	6	5	5	3	1	3	2	3
hand2	12	14	9	9	15	11	11	10	11	10	7	8	15	6	5
jogging	1	1	0	1	2	1	1	1	1	1	1	1	1	1	1
motocross	3	4	4	3	5	3	5	4	3	4	2	4	3	4	4
polarbear	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
skating	1	0	1	0	5	1	1	1	2	1	1	4	2	0	0
sphere	1	3	1	1	2	2	0	0	3	1	1	0	0	0	0
sunshade	4	3	3	4	10	3	3	3	3	3	3	4	4	0	0
surfing	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
torus	3	3	2	2	9	3	4	0	3	4	1	5	4	0	0
trellis	4	2	1	3	9	1	6	2	4	2	0	4	5	0	0
tunnel	2	3	1	0	7	2	0	0	1	0	2	2	0	0	0
woman	3	6	6	7	9	2	2	1	3	1	3	1	4	1	1
sum	**64**	**83**	**52**	**74**	**139**	**62**	**68**	**48**	**66**	**53**	**42**	**59**	**78**	**29**	**32**

Table4.3: Percentage of failures computed over sequences for each algorithm for all sequences

	Intensity	PCA	SIFT	PHOW	Sparse Features	Sparse SIFT	DAE	DAE_{color}	DAE_{gray}	$DAE_{gray,w/adapt}$	CNN	MIL	CT	DSST	SAMF
ball	0.003	0.007	0.003	0.008	0.005	0.005	0.003	0.002	0.003	0.002	0.003	0.002	0.002	0.002	0.002
basketball	0.004	0.003	0.001	0.007	0.007	0.004	0.003	0.004	0.000	0.003	0.003	0.003	0.001	0.001	0.000
bicycle	0.000	0.000	0.004	0.007	0.026	0.007	0.004	0.000	0.004	0.004	0.004	0.000	0.007	0.000	0.000
bolt	0.009	0.020	0.003	0.009	0.011	0.011	0.011	0.003	0.009	0.006	0.006	0.017	0.026	0.003	0.006
car	0.000	0.004	0.000	0.004	0.008	0.004	0.004	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000
david	0.003	0.001	0.000	0.001	0.004	0.003	0.000	0.001	0.000	0.000	0.000	0.000	0.001	0.000	0.000
diving	0.009	0.014	0.009	0.014	0.014	0.009	0.009	0.005	0.005	0.014	0.009	0.005	0.009	0.005	0.018
drunk	0.000	0.001	0.001	0.001	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
fernando	0.003	0.010	0.010	0.010	0.014	0.010	0.014	0.010	0.014	0.003	0.007	0.007	0.010	0.003	0.003
fish1	0.007	0.018	0.007	0.014	0.023	0.007	0.011	0.002	0.009	0.005	0.002	0.005	0.023	0.002	0.007
fish2	0.019	0.016	0.013	0.019	0.026	0.010	0.019	0.019	0.023	0.019	0.013	0.019	0.013	0.013	0.016
gymnastics	0.019	0.010	0.010	0.019	0.019	0.019	0.019	0.010	0.024	0.014	0.010	0.024	0.019	0.024	0.010
hand1	0.016	0.029	0.016	0.016	0.037	0.012	0.016	0.025	0.020	0.020	0.012	0.004	0.012	0.008	0.012
hand2	0.045	0.052	0.034	0.034	0.056	0.041	0.041	0.037	0.041	0.037	0.026	0.030	0.056	0.022	0.019
jogging	0.003	0.003	0.000	0.003	0.007	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
motocross	0.018	0.024	0.024	0.018	0.030	0.018	0.030	0.024	0.018	0.024	0.012	0.024	0.018	0.024	0.024
polarbear	0.000	0.000	0.000	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
skating	0.003	0.000	0.003	0.000	0.013	0.003	0.003	0.003	0.005	0.003	0.003	0.010	0.005	0.000	0.000
sphere	0.005	0.015	0.005	0.005	0.010	0.010	0.000	0.000	0.015	0.005	0.005	0.000	0.000	0.000	0.000
sunshade	0.023	0.017	0.017	0.023	0.058	0.017	0.017	0.017	0.017	0.017	0.017	0.023	0.023	0.000	0.000
surfing	0.000	0.000	0.000	0.000	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
torus	0.011	0.011	0.008	0.008	0.034	0.011	0.015	0.000	0.011	0.015	0.004	0.019	0.015	0.000	0.000
trellis	0.007	0.004	0.002	0.005	0.016	0.002	0.011	0.004	0.007	0.004	0.000	0.007	0.009	0.000	0.000
tunnel	0.003	0.004	0.001	0.000	0.010	0.003	0.000	0.000	0.001	0.000	0.003	0.003	0.000	0.000	0.000
woman	0.005	0.010	0.010	0.012	0.015	0.003	0.003	0.002	0.005	0.002	0.005	0.002	0.007	0.002	0.002
average	**0.009**	**0.011**	**0.007**	**0.010**	**0.018**	**0.009**	**0.010**	**0.007**	**0.009**	**0.008**	**0.006**	**0.008**	**0.010**	**0.005**	**0.005**

words, trackers are barely restricted to fit in a motion model and the condition of classifier update is not very strong, which decreases the overall performance of the framework setup.

Keeping the bias in the accuracy results in mind, we observe that the best accuracy result belongs to sparse SIFT features, which exploit auxiliary data for transfer learning. Success of fully-connected NN representations as well as CNN features also emphasizes the benefit of transfer learning for visual object tracking. However, performance of SIFT descriptors are better than sparse SIFT features in terms of robustness, although the latter is carefully hand-designed with great effort and requires computational power.

The best representation in terms of robustness is CNN features and the representation that led to the second least number of failures is the fully connected networks pretrained as color autoencoders, which reveals the power of hierarchical representations for visual object tracking.

Dimension reduction via PCA performs worse than the raw intensity value tracking on average, so, we can also conclude that generative models that does not adapt itself during tracking do not promise better performance. Results also indicate the power of raw intensity values within a tracking by detection scheme.

On the other hand, autoencoders, which can also be regarded as generative models, perform better than both intensity and PCA representation. It can be related to the power or hierarchical representations that try to capture the underlying reasons of the given data and therefore contain discriminative features. Adaptation of the whole network improves not only the accuracy but also the robustness of the tracker according to the performance difference of DAE_{gray} and $\text{DAE}_{gray,w/adapt}$. Results clearly show that inclusion of color information improves the performance of a tracker with feed-forward neural networks, since there is a considerable change in number of failures given in Table 4.2. The network with higher capacity that is trained on a larger dataset performs very similar to the small-capacity network in terms of both tracking accuracy and robustness.

Sparse representations result in the worst performance regarding both accuracy and robustness. MAP estimate in motion model as well as the normalization of resultant sparse vector has been observed to improve the results; however, detailed work and tests on sparse representations may be regarded as a future work.

CHAPTER 5

CONCLUSION

5.1 Summary

In this thesis, utilization of hierarchical representations for visual object tracking problem is investigated. A brief survey of visual object tracking algorithms has revealed the variety of approaches and evaluation methodologies, which indicates that the topic is an active research area. Thus, investigation of a rather primitive but very important problem as representation is expected to facilitate finding the future work directions.

Until last decade, hand-crafted object representations have been utilized with a great success owing to human effort and expertise. However, availability of more strong and powerful hardware as well as large-scale data sets has contributed to the recent rebirth of fairly old neural networks. Hierarchical architectures started to exploit the underlying structure of large datasets as a virtue of better understanding the optimization problem related to this complex computational modules. In this thesis, we not only presented some of the recent deep learning algorithms that achieved state of the art for a wide range of research areas, but also introduced the basics of multi-layer feed-forward neural networks, autoencoder and convolutional neural networks.

Despite the success of hierarchical representations in object classification tasks, application of deep learning algorithms to visual object tracking problem is not straightforward. There have been a few works that focus on the integration of these two important research problems, since generic online tracking algorithms

have only one labeled sample, initial frame, which is supposed to be processed as soon as possible. On the other hand, deep architectures that consist of a huge number of adjustable parameters require large number of examples for tuning the internal structure. The first thing that can be considered to meet these two problems is transfer learning, which regards the generalization capability of representations as an opportunity to apply them for different tasks. Thus, we experimented with autoencoders that are trained on auxiliary data and pre-trained convolutional neural networks for the representation of objects.

These deep architectures are experimented within a tracking by detection scheme as an object representation module. Experiments are realized using a recent challenge toolkit and the results are compared with some hand-crafted representations employed in the same tracking scheme. Results prove the power of hierarchical representations for visual object problem via transfer learning methods.

5.2 Conclusion

The findings of this thesis can be summarized as follows:

- Dynamics of visual tracking systems is not straightforward. Representation of an image with intensity values performed better than some hand-crafted and learned representations, and more interestingly, the overall performance of the system can be considered better than two algorithms with lowest ranks in 2014 VOT challenge.

- Adaptivity of a model that targets tracking problem is very crucial on the resultant performance. Adaptation of both representation and classifier in tracking by detection scheme are observed to improve the results.

- Color information improves the performance of a tracker based on tacking by detection scheme without sacrificing too much from computational simplicity.

- The representation that leads to the most robust tracker is CNN features, which exploits both the spatial and color information. Results with off the shelf features indicate the generalization ability of deep architectures. The success of CNN features may be improved further by fine-tuning, which is indicated by the results of different feed-forward neural networks.

- Although one of the strongest assertions of deep learning algorithms against the hand-crafted features is the fact that hand-crafted features require effort and expert knowledge, training of these architectures is also not a simple task. There are a lot of factors that directly affect the performance of the representation and the interrelated structure of these factors require time to understand the dynamics. Determination of capacity, loss function, optimization method and optimization parameters become more of an issue if the target task does not give an opportunity of validation, as in the tracking problem.

5.3 Future Work

This thesis work constitutes a small step into the application of deep learning to visual object tracking area; however, it is of a great importance since it leads to an initial understanding of two huge literatures. It is more than obvious that integration will not be straightforward; however, results show that even without a lot of engineering, hierarchical representations give promising results within one of the most common ways of visual object tracking.

The first future direction will probably concentrate on the adaptation of hierarchical representations to a more local appearance. Adaptation criterion as well as the way it is adapted will be critical. Tracking by detection scheme itself contains a lot of ambiguities like the determination of positive and negative samples according to a predicted classification result; therefore, integration of temporal information without a direct classification step may benefit more from deep architectures. Since tracking corresponds to sequential processing of data, recurrent neural networks may also be employed for visual tracking algorithms.

REFERENCES

[1] Object tracking using {SIFT} features and mean shift. *Computer Vision and Image Understanding*, 113(3):345 – 352, 2009. Special Issue on Video Analysis.

[2] Sparse coding based visual tracking: Review and experimental comparison. *Pattern Recognition*, 46(7):1772 – 1788, 2013.

[3] J. Aggarwal and Q. Cai. Human motion analysis: a review. In *Nonrigid and Articulated Motion Workshop, 1997. Proceedings., IEEE*, pages 90–102, June 1997.

[4] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, 2002.

[5] B. Babenko, M.-H. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE, 2009.

[6] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[7] E. Batı. Deep convolutional neural networks with an application towards geospatial object recognition.

[8] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.

[9] L. Bazzani, H. Larochelle, V. Murino, J.-a. Ting, and N. D. Freitas. Learning attentional policies for tracking and recognition in video with deep networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 937–944, 2011.

[10] Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[11] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.

[12] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[13] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

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

[15] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[16] Y. Boykov and M.-P. Jolly. Interactive organ segmentation using graph cuts. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2000*, pages 276–286. Springer, 2000.

[17] L. Cehovin, M. Kristan, and A. Leonardis. Robust visual tracking using an adaptive coupled-layer visual model. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(4):941–953, 2013.

[18] L. Čehovin, A. Leonardis, and M. Kristan. Visual object tracking performance measures revisited. *arXiv preprint arXiv:1502.05803*, 2015.

[19] T. F. Chan and L. A. Vese. Active contours without edges. *Image processing, IEEE transactions on*, 10(2):266–277, 2001.

[20] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

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

[22] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[23] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[24] M. Danelljan, G. Häger, F. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014.

[25] M. Denil, L. Bazzani, H. Larochelle, and N. de Freitas. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012.

[26] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[27] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[28] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2155–2162. IEEE, 2014.

[29] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[30] S. Fazli, H. M. Pour, and H. Bouzari. Particle filter based object tracking with sift and color feature. In *Machine Vision, 2009. ICMV'09. Second International Conference on*, pages 89–93. IEEE, 2009.

[31] M. Felsberg. Enhanced distribution field tracking using channel representations. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 121–128. IEEE, 2013.

[32] K. Fukushima. Neocognitron.

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

[34] K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[35] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[36] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer vision and image understanding*, 73(1):82–98, 1999.

[37] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[38] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *BMVC*, volume 1, page 6, 2006.

[39] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[40] S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 263–270. IEEE, 2011.

[41] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[42] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

[43] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

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

[46] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *Computer Vision–ECCV 2012*, pages 702–715. Springer, 2012.

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

[48] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[49] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[50] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[51] K. Ho. 41 up-to-date facebook facts and stats.

[52] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991. Advisor: J. Schmidhuber.

[53] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[54] D. H. HUBEL and T. N. WIESEL. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148:574–591, Oct. 1959.

[55] J. Håstad and M. Goldmann. On the power of small-depth threshold circuits. *computational complexity*, 1(2):113–129, 1991.

[56] R. Jagathishwaran, K. Ravichandran, and P. Jayaraman. A survey on face detection and tracking. 2014.

[57] A. Jain, J. Tompson, M. Andriluka, G. W. Taylor, and C. Bregler. Learning human pose estimation features with convolutional networks. *arXiv preprint arXiv:1312.7302*, 2013.

[58] X. Jia, H. Lu, and M.-H. Yang. Visual tracking via adaptive structural local sparse appearance model. In *Computer vision and pattern recognition (CVPR), 2012 IEEE Conference on*, pages 1822–1829. IEEE, 2012.

[59] J. Jin, A. Dundar, J. Bates, C. Farabet, and E. Culurciello. Tracking with deep neural networks. In *Information Sciences and Systems (CISS), 2013 47th Annual Conference on*, pages 1–5. IEEE, 2013.

[60] Z. Kalal, J. Matas, and K. Mikolajczyk. Pn learning: Bootstrapping binary classifiers by structural constraints. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 49–56. IEEE, 2010.

[61] A. Karpathy. Demo: toy 2d classification with 2-layer neural network.

[62] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014.

[63] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1725–1732. IEEE, 2014.

[64] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. P. Pflugfelder, G. Fernández, G. Nebehay, F. Porikli, and L. Cehovin. A novel performance evaluation methodology for single-target trackers. *CoRR*, abs/1503.01313, 2015.

[65] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Cehovin, G. Nebehay, T. Vojir, G. Fernandez, A. Lukezic, A. Dimitriev, A. Petrosino, A. Saffari, B. Li, B. Han, C. Heng, C. Garcia, D. Pangersic, G. Hager, F. S. Khan, F. Oven, H. Possegger, H. Bischof, H. Nam, J. Zhu, J. Li, J. Y. Choi, J.-W.

Choi, J. F. Henriques, J. van de Weijer, J. Batista, K. Lebeda, K. Ofjall, K. M. Yi, L. Qin, L. Wen, M. E. Maresca, M. Danelljan, M. Felsberg, M.-M. Cheng, P. Torr, Q. Huang, R. Bowden, S. Hare, S. Y. Lim, S. Hong, S. Liao, S. Hadfield, S. Z. Li, S. Duffner, S. Golodetz, T. Mauthner, V. Vineet, W. Lin, Y. Li, Y. Qi, Z. Lei, and Z. Niu. The visual object tracking vot2014 challenge results. In *Proceedings, European Conference on Computer Vision (ECCV) Visual Object Tracking Challenge Workshop*, Zurich, Switzerland, September 2014.

[66] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

[67] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[68] H. Larochelle. Lecture 3: Rétropropagation des gradients et optimisation. IFT 725 Lecture Notes.

[69] H. Larochelle. Lecture 6: Autoencodeurs. IFT 725 Lecture Notes.

[70] H. Larochelle. Multilayer neural networks. IFT 725 Lecture Notes.

[71] Y. LeCun. Learning processes in an asymmetric threshold network. In E. Bienenstock, F. Fogelman-Soulié, and G. Weisbuch, editors, *Disordered systems and biological organization*, pages 233–240, Les Houches, France, 1986. Springer-Verlag.

[72] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[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*, 1(4):541–551, 1989.

[74] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[75] H. Li, Y. Li, and F. Porikli. Deeptrack: Learning discriminative feature representations online for robust visual tracking. *CoRR*, abs/1503.00072, 2015.

[76] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, and A. van den Hengel. A Survey of Appearance Models in Visual Object Tracking. *ACM Transactions on Intelligent Systems and Technology*, March 2013.

[77] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *Computer Vision-ECCV 2014 Workshops*, pages 254–265. Springer, 2014.

[78] 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.

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

[80] L. Mastin. Neurons and synapses.

[81] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[82] X. Mei and H. Ling. Robust visual tracking using &# x2113; 1 minimization. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1436–1443. IEEE, 2009.

[83] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. J. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, et al. Unsupervised and transfer learning challenge: a deep learning approach. *ICML Unsupervised and Transfer Learning*, 27:97–110, 2012.

[84] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

[85] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

[86] F. Miller, A. Vandome, and M. John. *Inverse Transform Sampling*. VDM Publishing, 2010.

[87] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2).

[88] M. Olazaran. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659, 1996.

[89] Y. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44 vol.1, Nov 1993.

[90] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[91] D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, Oxford, UK, 1997.

[92] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

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

[94] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011.

[95] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[96] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

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

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

[99] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[100] 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)*, pages 1–42, April 2015.

[101] J. Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[102] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[103] L. Sevilla-Lara and E. Learned-Miller. Distribution fields for tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1910–1917. IEEE, 2012.

[104] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[105] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: an experimental survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(7):1442–1468, 2014.

[106] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[107] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[108] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[109] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[110] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[111] Y. Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[112] S. J. Thorpe and M. Fabre-Thorpe. Neuroscience. Seeking categories in the brain. *Science (New York, N.Y.)*, 291(5502):260–263, Jan. 2001.

[113] J. van de Weijer, C. Schmid, J. Verbeek, and D. Larlus. Learning color names for real-world applications. *Image Processing, IEEE Transactions on*, 18(7):1512–1523, July 2009.

[114] P. Vandergheynst and P. Frossard. Efficient image representation by anisotropic refinement in matching pursuit. In *Acoustics, Speech, and*

Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on, volume 3, pages 1757–1760 vol.3, 2001.

[115] P. Vincent. Machine learning from linear regression to neural networks. IFT3395 Lecture Notes.

[116] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[117] N. Wang, J. Wang, and D.-Y. Yeung. Online robust non-negative dictionary learning for visual tracking. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 657–664. IEEE, 2013.

[118] N. Wang and D.-Y. Yeung. Learning a deep compact image representation for visual tracking. In *Advances in Neural Information Processing Systems*, pages 809–817, 2013.

[119] Q. Wang, F. Chen, J. Yang, W. Xu, and M.-H. Yang. Transferring visual prior for online object tracking. *Image Processing, IEEE Transactions on*, 21(7):3296–3305, 2012.

[120] Q. Wang, F. Chen, J. Yang, W. Xu, and M.-H. Yang. Transferring visual prior for online object tracking. *Image Processing, IEEE Transactions on*, 21(7):3296–3305, 2012.

[121] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *Computer vision and pattern recognition (CVPR), 2013 IEEE Conference on*, pages 2411–2418. IEEE, 2013.

[122] J. Xiao, R. Stolkin, and A. Leonardis. An enhanced adaptive coupled-layer lgtracker++. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 137–144. IEEE, 2013.

[123] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[124] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

[125] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[126] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[127] K. Zhang, L. Zhang, and M.-H. Yang. Real-time compressive tracking. In *Computer Vision–ECCV 2012*, pages 864–877. Springer, 2012.

[128] S. Zhang, H. Yao, X. Sun, and X. Lu. Sparse coding based visual tracking: review and experimental comparison. *Pattern Recognition*, 46(7):1772–1788, 2013.

[129] S. Zheng, S. Jayasumana, Bernardino, Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. *CoRR*, abs/1502.03240, 2015.

[130] W. Zhong, H. Lu, and M.-H. Yang. Robust object tracking via sparsity-based collaborative model. In *Computer vision and pattern recognition (CVPR), 2012 IEEE Conference on*, pages 1838–1845. IEEE, 2012.

[131] L. Zhu, Y. Chen, A. Yuille, and W. Freeman. Latent hierarchical structural learning for object detection. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1062–1069. IEEE, 2010.