

Witold Pedrycz  
Shyi-Ming Chen *Editors*

# Deep Learning: Algorithms and Applications

# **Studies in Computational Intelligence**

Volume 865

## **Series Editor**

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland

The series “Studies in Computational Intelligence” (SCI) publishes new developments and advances in the various areas of computational intelligence—quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life sciences, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems, and hybrid intelligent systems. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution, which enable both wide and rapid dissemination of research output.

The books of this series are submitted to indexing to Web of Science, EI-Compendex, DBLP, SCOPUS, Google Scholar and Springerlink.

More information about this series at <http://www.springer.com/series/7092>

Witold Pedrycz · Shyi-Ming Chen  
Editors

# Deep Learning: Algorithms and Applications

*Editors*

Witold Pedrycz  
Department of Electrical  
and Computer Engineering  
University of Alberta  
Edmonton, AB, Canada

Shyi-Ming Chen  
Department of Computer Science  
and Information Engineering  
National Taiwan University of Science  
and Technology  
Taipei, Taiwan

ISSN 1860-949X

ISSN 1860-9503 (electronic)

Studies in Computational Intelligence

ISBN 978-3-030-31759-1

ISBN 978-3-030-31760-7 (eBook)

<https://doi.org/10.1007/978-3-030-31760-7>

© Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

Deep learning has entered a period of designing, implementing, and deploying intensive and diverse applications, which are now visible in numerous areas. Successful case studies became a consequence of the prudent, carefully drafted fundamental concept that transformed ways in such real-world problems are perceived, formalized, and solved at the increased level of machine-centered and automatic fashion. Central to all pursuits are algorithms that help realize the principles of deep learning in an efficient way. Algorithms deliver a sound alignment with the specificity of the applied nature of the practical problem by addressing the computing requirements and selecting/adjusting overall algorithmic settings.

The volume is composed of 11 chapters and reflects the wealth of algorithms of deep learning and their application studies in the plethora of areas including imaging, seismic tomography, power series forecasting, smart grids, surveillance, security, health care, environmental engineering, and marine sciences.

We would like to express our thanks to the Series Editor, Prof. Janusz Kacprzyk. He has always been enthusiastic and highly supportive of this project. We are indebted to the professionals at Springer; the team has made the overall production process highly efficient and completed in a timely manner.

Edmonton, Canada  
Taipei, Taiwan

Witold Pedrycz  
Shyi-Ming Chen

# Contents

<b>Activation Functions . . . . .</b>	1
Mohit Goyal, Rajan Goyal, P. Venkatappa Reddy and Brejesh Lall	
1 Introduction . . . . .	1
1.1 Drawbacks of Fixed Activation Functions . . . . .	4
2 Existing Activation Functions . . . . .	4
2.1 Linear Activation . . . . .	4
2.2 Binary Step Activation . . . . .	5
2.3 Sigmoid and Softmax . . . . .	6
2.4 Tanh (Hyperbolic Tan) . . . . .	8
2.5 The ReLU Family . . . . .	10
2.6 Softplus . . . . .	15
2.7 Maxout . . . . .	16
2.8 Swish Activation Function . . . . .	17
3 Comparison of Activation Functions . . . . .	18
4 Learning Activation Functions . . . . .	20
4.1 Learning Adaptive Piecewise Linear (APL) Activation Functions . . . . .	20
4.2 SLAF: Learning Non-linear Approximation of Activation Functions . . . . .	21
5 Conclusion . . . . .	27
6 Future Works . . . . .	28
References . . . . .	28
<b>Adversarial Examples in Deep Neural Networks: An Overview . . . . .</b>	31
Emilio Rafael Balda, Arash Behboodi and Rudolf Mathar	
1 Introduction . . . . .	32
1.1 Notation and Preliminaries . . . . .	33
2 Adversarial Perturbation Design . . . . .	34
2.1 White-Box Attacks . . . . .	37
2.2 Black-Box Attacks and Universal Adversarial Perturbations . . . . .	44

3	Theoretical Explanations of the Nature of Adversarial Examples . . . . .	47
3.1	Linearity Hypothesis and Curvature of Decision Boundaries . . . . .	49
3.2	Boundary Tilting and Other Explanations . . . . .	52
3.3	Feature Selection and No Free Lunch Theorems for Adversarial Robustness . . . . .	54
3.4	Generalization Bounds for Adversarial Examples . . . . .	56
4	Defenses Against Adversarial Attacks . . . . .	58
4.1	Obfuscated Gradients and Adversarial Training . . . . .	58
4.2	Robust Regularization . . . . .	59
5	Future Directions . . . . .	61
	References . . . . .	62
	<b>Representation Learning in Power Time Series Forecasting . . . . .</b>	67
	Janosch Henze, Jens Schreiber and Bernhard Sick	
1	Introduction . . . . .	68
2	Regression in Power Time Series Forecasting . . . . .	69
2.1	Renewable Power Time Series Forecasting . . . . .	70
2.2	Challenges of Power Time Series Forecasting . . . . .	71
3	Foundations of Representation Learning . . . . .	72
3.1	Traditional Dimensionality Reduction Techniques . . . . .	73
3.2	Deep Architectures for Latent Feature Extraction . . . . .	76
4	Evaluation of Representation Learning in Regression Tasks . . . . .	81
4.1	Evaluation of a Regression Model . . . . .	82
4.2	Evaluation of the Learned Feature Representation . . . . .	83
5	Representation Learning Applied in Power Time Series Forecasting . . . . .	85
5.1	The Power Time Series . . . . .	85
5.2	Representation Learning Experiments . . . . .	86
5.3	Principal Component Analysis for Feature Extraction . . . . .	88
5.4	Deep Architectures for Feature Extraction . . . . .	92
5.5	Fine-Tuning for Power Forecasting . . . . .	95
5.6	Key Insights . . . . .	98
6	Conclusion . . . . .	98
6.1	How to Build a Representation Learning Model . . . . .	99
	References . . . . .	99
	<b>Deep Learning Application: Load Forecasting in Big Data of Smart Grids . . . . .</b>	103
	Abdulaziz Almalaq and Jun Jason Zhang	
1	Introduction . . . . .	104
2	Background . . . . .	106
2.1	Electrical Load . . . . .	107
2.2	Load Forecasting . . . . .	107
2.3	Influential Factors of Load Forecasting Models . . . . .	109

3	Forecasting Modeling Issues in Smart Grids . . . . .	110
3.1	General Issues with Load Forecasting Modeling . . . . .	111
3.2	Traditional Load Forecasting Models . . . . .	111
3.3	Traditional Load Forecasting Models . . . . .	117
4	Solutions and Recommendations . . . . .	118
4.1	Guidelines and Solutions to Modeling Issues . . . . .	118
4.2	Case Study . . . . .	121
5	Conclusions and Future Trends . . . . .	125
	References . . . . .	126
	<b>Fast and Accurate Seismic Tomography via Deep Learning . . . . .</b>	129
	Mauricio Araya-Polo, Amir Adler, Stuart Farris and Joseph Jennings	
1	Introduction . . . . .	130
2	The Seismic Tomography Problem . . . . .	131
2.1	Seismic Data . . . . .	131
2.2	Seismic Tomography . . . . .	133
3	Seismic Tomography via Deep Learning . . . . .	135
3.1	Deep Neural Networks for Inverse Imaging Problems . . . . .	135
3.2	Velocity Semblance as Input Feature for Deep Networks . . . . .	138
4	Semblance-Based CNN Results . . . . .	139
4.1	Experimental Setup . . . . .	139
4.2	Quantitative Metrics . . . . .	141
4.3	Results and Analysis . . . . .	141
5	Industry Baseline: Full Waveform Inversion . . . . .	143
5.1	Industry VMB Methods . . . . .	143
5.2	Full Waveform Inversion . . . . .	144
5.3	Baseline Comparison Setup . . . . .	146
5.4	Experiments . . . . .	147
5.5	Results . . . . .	147
5.6	Discussion . . . . .	148
6	Feature Extraction-Free Results . . . . .	152
7	Conclusions . . . . .	154
	References . . . . .	154
	<b>Traffic Light and Vehicle Signal Recognition with High Dynamic Range Imaging and Deep Learning . . . . .</b>	157
	Jian-Gang Wang and Lu-Bing Zhou	
1	Introduction . . . . .	158
2	HDR Imaging Traffic Light Detection . . . . .	160
2.1	HDR Imaging . . . . .	160
2.2	Dark Images for Detecting Traffic Light Candidates . . . . .	161
2.3	Saliency Map Filtering . . . . .	163
2.4	Auto Exposure for Uncontrolled Illumination . . . . .	164
2.5	Region of Interest (ROI) . . . . .	165

3	Traffic Light Recognition with Deep Learning . . . . .	167
3.1	Dual Channel Mechanism . . . . .	167
3.2	Customized Convolutional Neural Network . . . . .	168
4	Temporal Trajectory Analysis . . . . .	170
5	Experimental Results . . . . .	172
5.1	Evaluation of Performance . . . . .	172
5.2	Comparison with State of the Art Algorithms . . . . .	173
6	HDR Imaging Vehicle Signal Recognition . . . . .	183
6.1	Related Approaches . . . . .	185
6.2	Two-Stage Vehicle Signal Recognition . . . . .	185
6.3	Vehicle Detection . . . . .	185
6.4	Brake Light Pattern and Recognition . . . . .	186
6.5	Experimental Results . . . . .	187
7	Conclusion and Future Work . . . . .	189
	References . . . . .	190

## **The Application of Deep Learning in Marine Sciences . . . . .** 193

Miguel Martin-Abadal, Ana Ruiz-Frau, Hilmar Hinz  
and Yolanda Gonzalez-Cid

1	Introduction . . . . .	194
2	Methodology . . . . .	195
3	Seagrass Segmentation . . . . .	197
3.1	Deep Learning Approach . . . . .	199
3.2	Experimental Framework . . . . .	202
3.3	Classification Results . . . . .	207
4	Jellyfish Detection and Identification . . . . .	213
4.1	Deep Learning Approach . . . . .	214
4.2	Experimental Framework . . . . .	218
4.3	Results . . . . .	223
5	Conclusions . . . . .	226
	References . . . . .	227

## **Deep Learning Case Study on Imbalanced Training Data for Automatic Bird Identification . . . . .** 231

Juha Niemi and Juha T. Tanttu

1	Introduction . . . . .	232
2	The System . . . . .	233
3	Related Work . . . . .	234
4	Data . . . . .	235
4.1	Data Augmentation . . . . .	237
4.2	Grouping Data . . . . .	238
5	Classification . . . . .	240
5.1	Hyperparameter Selection . . . . .	241
5.2	Feature Extraction . . . . .	241

5.3	Tests for Deeper CNN Model . . . . .	244
5.4	Dealing with Imbalanced Data . . . . .	244
5.5	Hybrid of Hierarchical and Cascade Model . . . . .	246
5.6	Top-Level Classification . . . . .	248
5.7	Classification of Waterfowl . . . . .	252
5.8	Classification of Gulls and Terns . . . . .	253
6	Results . . . . .	256
7	Discussion . . . . .	260
	References . . . . .	261
	<b>Deep Learning for Person Re-identification in Surveillance Videos . . . . .</b>	263
	Swathi Jamjala Narayanan, Boominathan Perumal, Sangeetha Saman and Aditya Pratap Singh	
1	Introduction . . . . .	264
2	Preliminaries of Deep Neural Networks . . . . .	265
3	Person Re-identification Datasets . . . . .	280
4	Deep Learning Architectures for Person Re-identification . . . . .	280
5	Experimental Setup . . . . .	289
6	Conclusions and Future Work . . . . .	292
	References . . . . .	293
	<b>Deep Learning in Gait Analysis for Security and Healthcare . . . . .</b>	299
	Omar Costilla-Reyes, Ruben Vera-Rodriguez, Abdullah S. Alharthi, Syed U. Yunas and Krikor B. Ozanyan	
1	Introduction . . . . .	300
2	Gait Analysis Review . . . . .	301
2.1	Non-wearable Sensors . . . . .	303
2.2	Wearable Sensors . . . . .	305
2.3	A Review of Floor Sensor Systems and Datasets for Gait Analysis . . . . .	306
3	Deep Learning for Gait Analysis . . . . .	307
3.1	Convolutional Neural Networks . . . . .	309
4	Deep Learning in Healthcare: A Case Study in Dual-Tasks . . . . .	311
4.1	Aims and General Method . . . . .	311
4.2	Background . . . . .	312
4.3	Methodology . . . . .	312
4.4	Experiments for Age-Related Classification . . . . .	313
4.5	Spatio-Temporal Deep Learning Model . . . . .	314
4.6	Results for Age-Related Differences . . . . .	315
4.7	Analysis of Experiments Three and Seven . . . . .	317
4.8	Discussion . . . . .	317
4.9	Future Directions . . . . .	320

5	Deep Learning in Security: A Case Study in Biometrics . . . . .	321
5.1	Aims and General Method . . . . .	321
5.2	Footstep Data as a Biometric . . . . .	322
5.3	Deep Residual Network Model . . . . .	323
5.4	Spatial and Temporal Architectures . . . . .	325
5.5	Verification System Evaluation . . . . .	326
5.6	Results . . . . .	326
5.7	Home Scenario: Benchmark B3 . . . . .	327
5.8	Discussion . . . . .	327
6	Conclusions . . . . .	329
	References . . . . .	330
<b>Deep Learning for Building Occupancy Estimation Using Environmental Sensors . . . . .</b>		335
Zhenghua Chen, Chaoyang Jiang, Mustafa K. Masood, Yeng Chai Soh, Min Wu and Xiaoli Li		
1	Introduction . . . . .	336
2	Literature Review . . . . .	337
3	Methodology . . . . .	338
3.1	Overview . . . . .	338
3.2	Convolutional Operation . . . . .	340
3.3	Deep Bi-directional LSTM . . . . .	341
3.4	Occupancy Inference Layers . . . . .	344
3.5	Training Process of the CDBLSTM . . . . .	344
4	Evaluation Results . . . . .	345
4.1	Data Collection . . . . .	346
4.2	Evaluation Setup . . . . .	347
4.3	Evaluation Results . . . . .	348
4.4	HyperParameters . . . . .	351
4.5	The Impact of Noise . . . . .	352
4.6	Generalization Performance . . . . .	353
4.7	Additional Evaluation with Data from Another Environment . . . . .	354
5	Conclusion . . . . .	355
	References . . . . .	355
	<b>Index . . . . .</b>	359

# Activation Functions



Mohit Goyal, Rajan Goyal, P. Venkatappa Reddy and Brejesh Lall

**Abstract** Activation functions lie at the core of deep neural networks allowing them to learn arbitrarily complex mappings. Without any activation, a neural network will only be able to learn a linear relation between input and the desired output. The chapter introduces the reader to why activation functions are useful and their immense importance in making deep learning successful. A detailed survey of several existing activation functions is provided in this chapter covering their functional forms, original motivations, merits as well as demerits. The chapter also discusses the domain of learnable activation functions and proposes a novel activation ‘SLAF’ whose shape is learned during the training of a neural network. A working model for SLAF is provided and its performance is experimentally shown on XOR and MNIST classification tasks.

**Keywords** Activation functions · Neural networks · Learning deep neural networks · Adaptive activation functions · ReLU · SLAF

## 1 Introduction

Neural Networks (NNs) are powerful information processing tools and can learn data representation [1] implicitly and efficiently. As a result, these networks have been shown to give excellent performance on tasks like statistical pattern recognition,

---

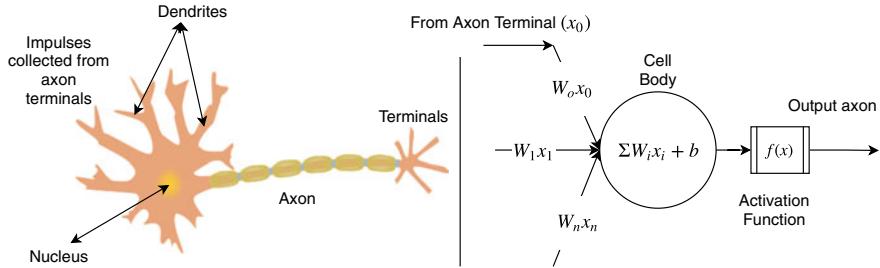
Mohit Goyal and Rajan Goyal have Equally Contributed.

---

M. Goyal (✉) · R. Goyal · P. Venkatappa Reddy · B. Lall  
Department of Electrical Engineering, Indian Institute of Technology Delhi, Delhi, India  
e-mail: [goyal.mohit999@gmail.com](mailto:goyal.mohit999@gmail.com)

R. Goyal  
e-mail: [rajangoyal910@gmail.com](mailto:rajangoyal910@gmail.com)

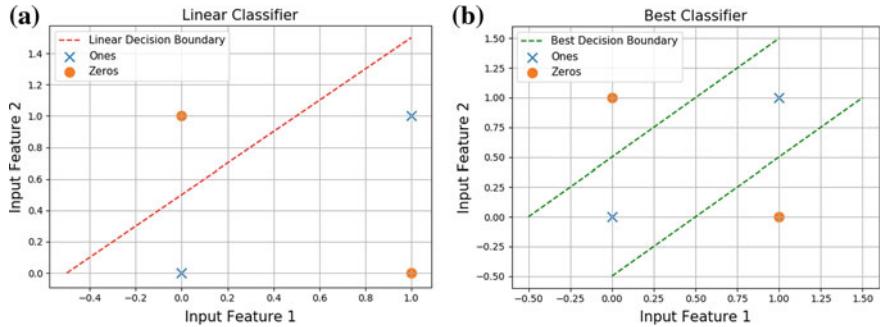
P. Venkatappa Reddy  
Electronics and Communication Engineering, Vignan’s Foundation for Science,  
Technology & Research, Guntur, India



**Fig. 1** Cartoon relating neuron and a neural network component

classification, time series prediction, etc., [2, 3]. NNs are functionally inspired from working of a human brain. As there are trillions of neurons in a brain heavily interconnected to each other, so are in NNs, providing path for the information to flow through. Similar to human beings, NNs can also learn from examples and can make predictions/decisions based on the observed trends. Analogous to firing of only specific neurons in the brain, only some of the nodes in any NN's hidden layer are activated in response to an input stimuli. This firing of neuron comes from the term action potential [4]. It plays an important role in cell to cell communication by assisting the propagation of signals along the neuron's axon. A very simplistic model is shown in Fig. 1 (derived from [5]). A neuron receives signals ( $x_0$ ) from the other neuron's axon terminals, which are collected through dendrites. These signals undergo multiplicative interaction in dendrites ( $W_0x_0$ ) based on the synaptic strength ( $W_0$ ). Note that  $W$ 's can be excitatory when the  $W$ 's are positive as well as inhibitory when  $W$ 's are negative. If the final sum obtained ( $\sum W_i x_i + b$ ) is greater than the threshold, then the neuron fires. This frequency of firing is modeled by an activation function in NN. This model doesn't generalize to all kinds of neurons and takes certain assumptions which might not be satisfied in actual neurons. Interested readers can refer to [6] for more insights.

One of the early form of neural networks is **Perceptron**. It was developed at Cornell Aeronautical Laboratory by Frank Rosenblatt [7] for image classification task. The weights of the network were stored in physical potentiometers, and the learning occurred with the help of electric motors which would update these weights during the training phase of the neural network. Technically, the perceptron was designed to learn a mapping from input space, i.e., pixels of image to a binary space corresponding to the class that image belongs to. To realize this mapping, a threshold function with only two possible outcomes was required. Hence, the step function was used at the output which gave a unit value for positive inputs and zero for negative input values. Later, when the stochastic gradient descent [8] became popular, a differentiable and continuous approximation of step function called sigmoid took its place. It is a doubly saturating non linear function whose output is bounded between zero and one. This gave rise to logistic regression which became the de facto for classification tasks. Unlike the step function which was a hard classifier, the output of sigmoid can be interpreted as the probability of belonging to a particular class.



**Fig. 2** **a** Linear decision boundary for XOR problem **b** Best decision boundary for XOR problem

Though simple and easy to use, logistic regression exhibits a major downside as well when the classes are not linearly separable. To better understand linear separability of classes, let's look at the XOR (exclusive OR) classification problem. It is easy to see that it is not linearly separable, (Fig. 2) and a linear classifier would only classify seventy five percent of the points correctly (Fig. 2a). On the other hand, the perfect decision boundary would have to be non linear. Figure 2b shows one possible boundary for XOR problem defined by two straight lines dividing the complete features space into three regions. The middle region belongs to class, labelled as one and the other two belong to the class, labelled as zero. To increase the representation capability of NNs such that they can learn non linear boundaries, a hidden layer is introduced between input and output which readers have learned in the earlier parts of the book. This provides the ability to learn more complex and highly non linear functions, to the neural networks (NN) which in the chapter is informally called as an increase in the capacity of NNs. It is well known that increasing width and depth increase the representation capacity of NN in general. Hence, it becomes important to ask this question, ‘does the choice activation affect the relationship between capacity and width or depth of a deep neural network (DNN)?’. Universal function approximation theorem says that we can learn any function considering a wide enough architecture with any of the accepted activations. But does there exist a better activation function, and if it does, how do we characterize it?

Another objective of this chapter is to allow readers to not only choose the correct activation function, but rather understand the reasons as to why a certain activation might perform better. It is well known that (Long Short Term Memory) LSTMs, variants of recurrent neural networks, use hyperbolic tan as the activation function and sigmoid activation function for gating mechanism. Since sigmoid has significance in terms of its interpretability as probability, it is intuitive to use it for gating or binary classification problems. In the later part of chapter, we will understand that sigmoid suffers from vanishing gradient problem. Hyperbolic tan on the other hand gives convergence in lesser iterations due to its mathematical properties which is why it is used in LSTMs over sigmoid. Such properties are important to understand for a researchers developing new architectures and activations.

## 1.1 Drawbacks of Fixed Activation Functions

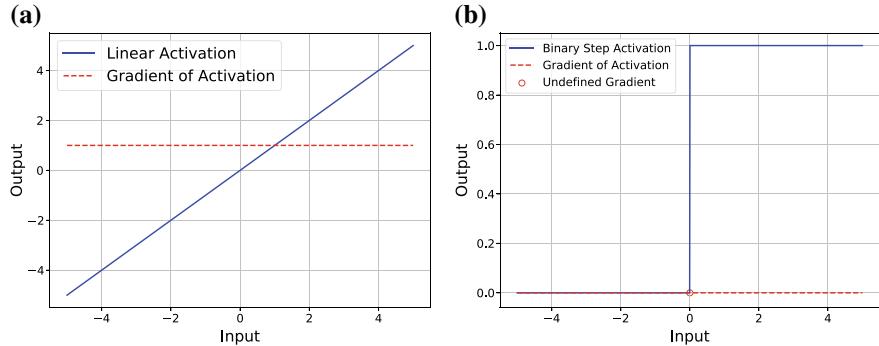
Tuning the width and depth of the network, applying regularization penalties [9, 10] and using normalization methods [11] have been shown to help in improving the performance of Deep Neural Networks (DNNs). Another important parameter that affects the performance of NNs is the type of activation function. As we will in the later parts of the chapter, no two activations performing equivalently even with same architectures, it might become critical to choose the right activation function. Hence, it can be seen as a hyper parameter which can be tuned specifically for the task and data set. One possible way is to try all existing activation functions on the architecture and compare their performance. However, this will be impractical when we consider using different activation function at each layer, or maybe at even each node. Another problem that one can be pointed out in fixed activation functions is that they will inevitably end up providing some non-linearity in the architecture. For example, it is impossible to learn identity operation using neural networks. Keeping the activations as fixed, it is impossible to recover input without loss of information. This can be seen as an important issue with auto encoders [12]. Even with high dimensional encoding, the resulting output is always lossy. One might say that, had the activation function been linear, this problem wouldn't occur. These factors serve as motivation for using and studying adaptive/learnable activation functions. In the last section of the chapter, we will see the design of such activation functions and understand their practical importance.

## 2 Existing Activation Functions

In this section, we will try to understand various activation functions present in the literature. We will study how they were proposed, why are they advantageous and how newer activations, developed onto them. Though we are far from fully understanding what actually happens inside a neural networks, but mathematical analysis of these non linearities can provide proper basis and intuition of what might be happening inside neural networks. One must understand activation functions not just as a tool to make NNs work but also their theoretical implications.

### 2.1 Linear Activation

Neural networks equipped with Linear Activation functions are generally called as linear neural networks. They have been widely studied to systematically study the learning dynamics of deep learning [13, 14]. They have analytic importance and are useful to understand the nature of neural networks. Linear neural networks with multiple hidden layers are equivalent to neural networks of single hidden layer.



**Fig. 3** **a** Linear activation function and its gradient **b** Binary step activation function and its gradient

They are used for linear regression tasks and a closed form solution exists for such networks. Linear activation functions are defined as

$$y = x, \quad (1)$$

where,  $x$  is the input to activation function and  $y$  is its output. It is clear from Eq. (1) that the gradient of linear activation functions is unity. Figure 3a shows the activation function and its gradient pictorially.

## 2.2 *Binary Step Activation*

A binary step function is similar to a threshold function that can be used for the purpose of binary classification tasks [15, 16]. Although, this function is quite old, it still has historical importance in machine learning. It is used in classification tasks done with signal processing methods. It is mathematically defined (for threshold at 0) as:

$$B.\ Step(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases}$$

Its gradient can be written as:

$$\frac{d(B.\ Step(x))}{dx} = \begin{cases} \text{Undefined} & \text{if } x = 0 \\ 0 & \text{if otherwise} \end{cases}$$

This function is non-differentiable at zero (or generally threshold) and has zero gradient at all other points. Hence, it is used in conjunction with stochastic gradient descent. This led to discovery of a smoother, differentiable approximation known as

sigmoid through which gradients could flow backwards. Let's see what sigmoid and its multi-class variant look like.

### 2.3 Sigmoid and Softmax

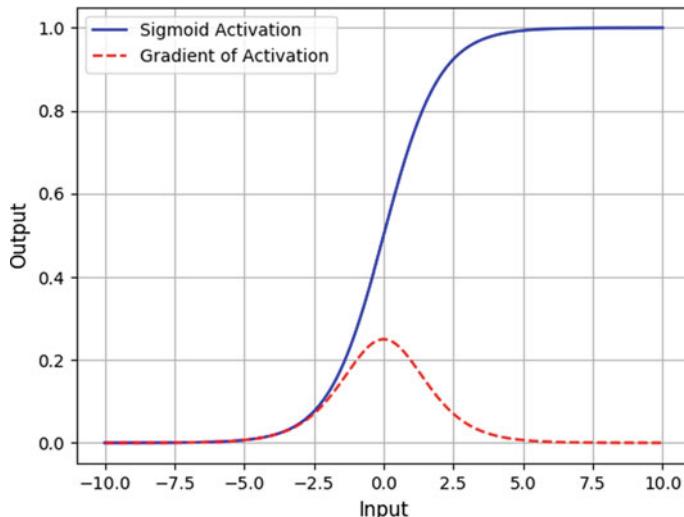
Sigmoid was originally designed for binary classification but now has wide application in tasks related to attention models, and bounded output regression as well [17, 18]. Mathematically, sigmoid is defined as:

$$\text{sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

The output of sigmoid lies in the range  $[0, 1]$  and hence, it can be interpreted as probability. Its gradient is interestingly easy to compute as it can be expressed in terms of the original activation itself:

$$\frac{d(\sigma(x))}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x) \cdot (1 - \sigma(x))$$

The major problem which we will discuss below with sigmoid is the vanishing gradient problem. The gradient of sigmoid activation function as shown in Fig. 4 vanishes away from zero and is upper bounded by value 0.25. This characteristic makes the training of deep neural networks slower when all its activations are sigmoid.



**Fig. 4** Sigmoid activation function and its corresponding gradient

Before we discuss performance details of sigmoid, one would ask this question. Why would anyone specifically use only this form of sigmoid? There are multiple ways to approximation binary step function. So why this? To answer that, we can look at the argument proposed in [19].

Consider a generative model for generating data denoted by random variable  $X$ , where,  $x$  (one realization of  $X$ ) can either belong to class  $\mathcal{C}_1$  or  $\mathcal{C}_2$ . This model is expressed by the prior probability distribution  $P(\mathcal{C}_1), P(\mathcal{C}_2)$  and class conditional probability density functions  $p(x|\mathcal{C}_1), p(x|\mathcal{C}_2)$ . Now, the posterior estimates on probability of an input belonging to either of the class and, can be written with the help of Bayes' theorem as:

$$\begin{aligned} p(\mathcal{C}_1|x) &= \frac{p(x|\mathcal{C}_1) \cdot P(\mathcal{C}_1)}{p(x|\mathcal{C}_1) \cdot P(\mathcal{C}_1) + p(x|\mathcal{C}_2) \cdot P(\mathcal{C}_2)} \\ &= \frac{1}{1 + \exp(-a)} \\ &= \sigma(a) \end{aligned} \quad (2)$$

where,  $a = \ln(\frac{p(\mathcal{C}_1|x)}{p(\mathcal{C}_2|x)})$ , and is frequently referred to as ‘logits’.

The above idea is extended to write the posterior probability for multiple classes (let's say  $K$  classes  $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K)$ ). This gives rise to the softmax activation function [19]:

$$\begin{aligned} p(\mathcal{C}_k|x) &= \frac{p(x|\mathcal{C}_k) \cdot P(\mathcal{C}_k)}{\sum_{i=1}^K p(x|\mathcal{C}_i) \cdot P(\mathcal{C}_i)} \\ &= \frac{\exp(-z_k)}{\sum_{i=1}^K \exp(-z_i)} \\ &= \text{softmax}(\bar{z}) \end{aligned} \quad (3)$$

where,  $\bar{z}$  is a  $k$ -dimensional vector containing  $K$  logits, one for each class. This wraps up the two most important activation functions sigmoid and softmax. The gradient of softmax can also be written in terms of original function itself.

$$\frac{d(\text{softmax}(z_i))}{dz_j} = \frac{d(s(z_i))}{dz_j} = \begin{cases} s(z_i) \cdot (1 - s(z_i)) & \text{if } i = j \\ -s(z_i) \cdot s(z_j) & \text{if } i \neq j \end{cases} \quad (4)$$

Given such important meaning associated with sigmoid activation functions, their use inside hidden layer was initially a heuristic (inspired biologically). Later, it was found that sigmoid led to vanishing gradient problems. For cascaded sigmoid activations across  $m$  hidden layers of one node each (calling the outer most layer as  $h_1$ , its activated output as  $o_1$  and the incoming weight of the edge as  $w_1$ , such that  $h_k = w_k * o_{k+1}$  and  $o_k = \sigma(h_k)$ ), the corresponding gradient at  $k$ th layer is:

$$\frac{do_1}{dh_k} = \frac{do_1}{dh_1} \cdot w_1 \cdot \dots \cdot \frac{do_{k-1}}{dh_{k-1}} \cdot w_{k-1} \cdot \frac{do_k}{dh_k} \quad (5)$$

Now, each  $\frac{d\phi_i}{d\theta_i}$  is derivative of sigmoid which is upper bounded by a value of 0.25. These successive multiplications lead to a smaller and smaller value in shallower layers. This makes the learning slower for the shallower parameters. This disadvantage of sigmoid activation functions can be overcome by using hyperbolic tanh activation function. It has an interesting mathematical properties even its characteristics are close to sigmoid activation function.

## 2.4 Tanh (Hyperbolic Tan)

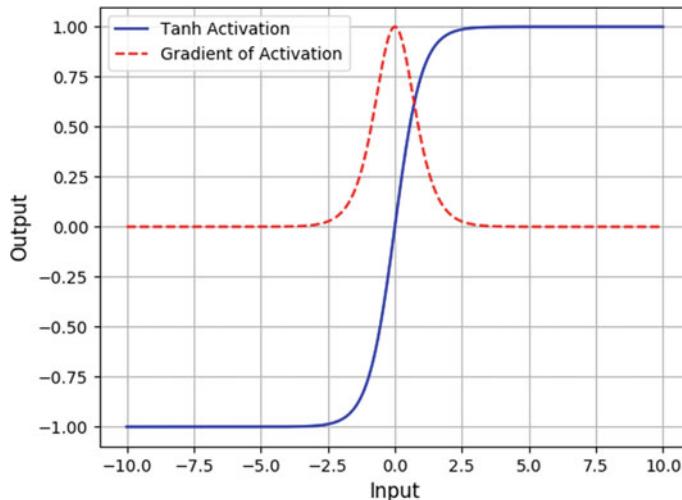
Hyperbolic tan is another activation function that can also be called as symmetric sigmoid [20]. It is a zero centered, doubly saturating activation function (saturating away from zero in both directions). It is mathematically defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (6)$$

and its gradient can be written as

$$\frac{d(\tanh(x))}{dx} = \frac{4}{(e^x + e^{-x})^2} = 1 - \tanh^2(x) \leq 1.0 \quad (7)$$

The above equations show that the gradient of tanh is upper bounded by one, which is four times that of sigmoid activation function. Therefore, due to larger gradients, symmetric sigmoids are often seen to converge faster (Fig. 5).



**Fig. 5** Tanh activation function and its corresponding gradient

Another property which contributes to superior performance of tanh compared to sigmoid activation function would be its zero centered nature. This means that if we uniformly sample points from  $\mathbb{R}$ , then apply tanh function on them, the output mean would be theoretically zero. The argument from [21] given below will help us to understand how this leads to the faster training of neural networks.

**Proposition 1** Consider a non linearity or activation function, denoted as  $f(\cdot)$ , being used in a neural network, trained with stochastic gradient descent. If the activation function is not zero centered, or  $\sum_{x \in \mathbb{R}} f(x) \neq 0$ , then the activation function will result in biased gradient and hence network will train slowly and reach optima in more iterations.

*Proof* Consider a multi-layered neural network with  $N$  hidden layers, with non linear vector transformation defined as  $F(\cdot)$ . This transformation takes in a vector, and applies an activation function  $f(\cdot)$  on its each component. The relation between  $(n - 1)$ th layer (with  $k$  inputs) of neural network to  $n$ th layer (with  $r$  inputs) can be written as:

$$Y_n = W_n X_{n-1} \quad (8)$$

$$X_n = F(Y_n) \quad (9)$$

where,  $W_n$  is the weight matrix of shape  $(r \times k)$ , and  $X_n$  is the  $n$ th hidden layer after applying activation  $F(\cdot)$  on  $Y_n$ . The derivatives can be calculated from the following recurrence relation.

$$\frac{\partial E^p}{\partial y_n^i} = f'(y_n^i) \frac{\partial E^p}{\partial x_n^i} \quad (10)$$

$$\frac{\partial E^p}{\partial x_{n-1}^i} = \sum_{j=1}^k w_n^{i,j} \frac{\partial E^p}{\partial y_n^j} \quad (11)$$

$$\frac{\partial E^p}{\partial w_n^{i,j}} = x_{n-1}^j \frac{\partial E^p}{\partial y_n^i} \quad (12)$$

Here,  $x^i, y^i, w^{i,j}$  represent the components of vectors  $X, Y$ , and matrix  $W$  respectively and  $E$  is the objective functions. In case where  $f(\cdot)$  is sigmoid activation function,  $X_{n-1}$  will always be positive. From Eq.(12), it can be easily inferred that  $w_n^{i,j} > 0 \quad \forall i, j \in \{(1, \dots, r), (1, \dots, k)\}$  if the partial derivative  $\frac{\partial E^p}{\partial y_n^i} > 0 \quad \forall i \in \{1, \dots, k\}$ . Therefore, gradients for the matrix defined as,  $w_n^{i,j} \quad \forall i, j \in \{(1, \dots, r), (1, \dots, k)\}$  will only point in the first quadrant or the third quadrant, where,  $r = 1$  and  $k = 2$ . To change the direction of this row vector the path followed by stochastic gradient descent is a zig-zag only. More generally, this effect is seen when the activation function is not zero centered. Generally, activation functions are biased to either positive or negative values. This will be translated to the gradients eventually, resulting in a longer convergence time. Hence, it is preferable to use zero centered activation

functions. This nature can be explicitly provided by using normalization techniques, which make activations zero centered at each layer where normalization is applied.  $\square$

Hyperbolic tan is a zero centered activation function and has major application in recurrent neural networks, where the problem of vanishing gradient is much more prominent. One should take note that if the gradient of an activation function is more than one, this would result in a phenomenon termed as exploding gradients [22]. This leads to instability during training and never achieves steady state convergence. Techniques such as gradient clipping [22] ensure that the gradient is appropriately scaled to lower values in events of gradient explosion.

## 2.5 The ReLU Family

ReLU is one of the most popular and widely used activation function. Many versions of the ReLU activation function have been proposed. Below we discuss the activation functions belonging to this family.

### 2.5.1 ReLU

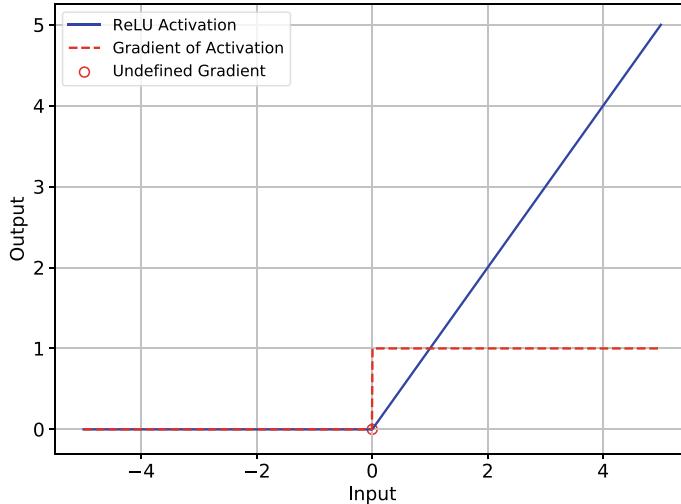
ReLU was proposed in [23, 24], and was shown to stabilize the feedback in analogue circuits. They proved that under certain conditions, networks equipped with ReLU non linearity would always converge to a steady state. Mathematically ReLU is defined as,

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (13)$$

Its gradient is given as,

$$\frac{d(\text{ReLU}(x))}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0 \end{cases} \quad (14)$$

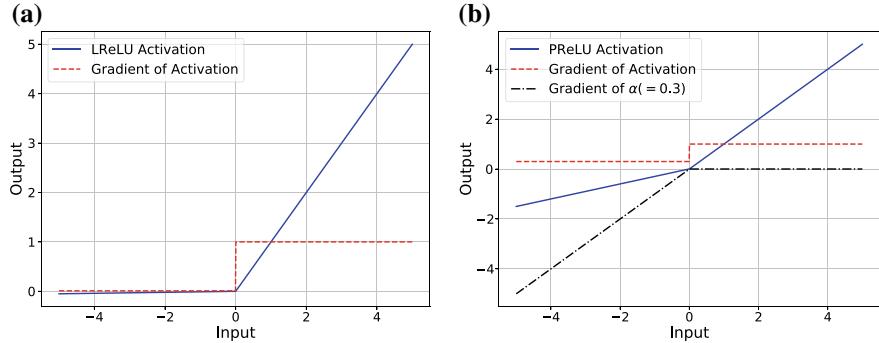
ReLU is a continuous function and differentiable at all points except at zero. It might seem that ReLU will not be compatible to stochastic gradient descent, However, there is only a infinitesimally small probability of landing up to an actual zero input to this activation function, given random initialization of architectures as well as presence of bias term before applying activation. In practice, ReLU is the most widely adopted activation function. There are several reasons that provide advantage to the ReLU activation (Fig. 6).



**Fig. 6** ReLU activation function and its corresponding gradient

1. **Gradient stability:** From Eq. (5), it is easy to see that no matter how deep we go, the gradient of ReLU (when input is positive), is always one and hence it will not vanish. A more generic formulation can be written using Eqs. (10), (11), (12). Hence, the case where the gradient of ReLU dies will happen when all of the ReLU outputs are zero across the hidden layer.
2. **Computationally cheaper:** Other activation functions require evaluation of exponents and performing division operations. However, ReLU is a simpler because it only requires a max operation to generate output. The same argument is valid for the gradient calculation. This is another reason, ReLU is preferable over other complex activation functions.
3. **Sparsity:** ReLUs result in sparsity in layers as if the input becomes zero, a connection becomes irrelevant to the model. This allows for analyzing the features, or variables which are important. However, if it actually makes the generalization or performance better is still a question.

In contrast to the above advantages of ReLU, there exists some drawbacks as well. ReLU can result in dead neurons, if the output of a particular activation becomes zero, then its gradient might die forever. Since the gradient flowing backward would also be zero, it might be possible that a neuron which could have contributed greatly, might never recover from that stage. Another problem with ReLU is its non-zero centric nature, because of which the activations are biased to being positive. Had the ReLU been zero centered, it certainly would have accelerated the training.



**Fig. 7** **a** LReLU and its corresponding gradient. **b** PReLU with  $\alpha = 0.3$  and its corresponding gradient

### 2.5.2 Leaky ReLU

A new activation function called leaky ReLU [25] is used to avoid dying problem in ReLU. The major disadvantage with ReLU is that it saturates at zero whenever it is not activated. This leads to zero gradient and leads to slower training. To solve this issue, a leak is added to the activation instead of hard zero. Mathematically LReLU is defined as:

$$LReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{if } x < 0 \end{cases} \quad (15)$$

This leak helps to increase the range of the ReLU. The gradient of the leaky ReLU is obtained as

$$\frac{dLReLU(x)}{dx} = \begin{cases} 1 & \text{if } x \geq 0 \\ 0.01 & \text{if } x < 0 \end{cases} \quad (16)$$

The characteristics of Leaky ReLU and its gradient are shown in Fig. 7a.

### 2.5.3 PReLU (Parametric ReLU)

Parametric rectified linear unit (PReLU) is a new generalization of ReLU introduced in [26]. It was originally defined for convolutional neural networks (CNNs). The PReLU is:

$$PReLU(x_i) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \alpha_i x_i & \text{if } x_i < 0 \end{cases} \quad (17)$$

where,  $\alpha_i$  is a parameter that is learned during training. Here, the subscript  $i$  refers to the  $i$ th channel in the convolutional neural network. For each feature map,  $\alpha_i$  is shared, thereby reducing the chance of over-fitting. LReLU uses  $\alpha_i = 0.01$ , but PReLU

adaptively learns the slope of the negative part. There's a trade off in computational cost when using PReLU with the motivation to learn better and specialized activation. The gradient of the PReLU is

$$\frac{dPReLU(x_i)}{dx_i} = \begin{cases} 1 & \text{if } x_i \geq 0 \\ \alpha_i & \text{if } x_i < 0 \end{cases} \quad (18)$$

The characteristics of PReLU and its gradient are shown in Fig. 7b. The update formulas of parameter  $\alpha_i$  are derived from chain rule. To write the update rules for the parameters  $\alpha_i$ , consider the equation below,

$$\frac{\partial E}{\partial \alpha_i} = \sum_{x_i} \frac{\partial E}{\partial PReLU(x_i)} \frac{\partial PReLU(x_i)}{\partial \alpha_i} \quad (19)$$

where,  $E$  represents the objective function. The term  $\frac{\partial E}{\partial PReLU(x_i)}$  is the gradient back propagated from the deeper layers. The gradient of the activation with  $\alpha_i$  is given by

$$\frac{\partial PReLU(x_i)}{\partial \alpha_i} = \begin{cases} 0 & \text{if } x_i > 0 \\ x_i & \text{if } x_i \leq 0 \end{cases} \quad (20)$$

The gradient of  $\alpha$ , when it is shared across both channels as well as feature maps is

$$\frac{\partial E}{\partial \alpha} = \sum_i \sum_{\alpha_i} \frac{\partial E}{\partial PReLU(x_i)} \frac{\partial PReLU(x_i)}{\partial \alpha} \quad (21)$$

The authors of PReLU activation function suggest using momentum optimizer for updating parameters  $\alpha_i$ . Hence, the update at  $n$ th iteration can be given as

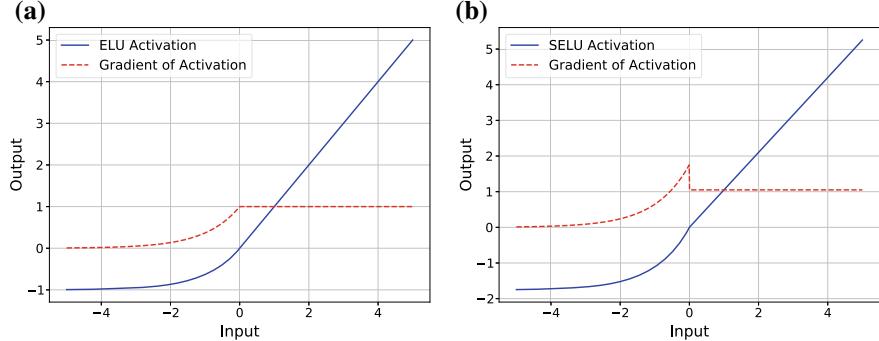
$$\delta \alpha_i^n = \mu \delta \alpha_i^{n-1} + \epsilon \frac{\partial E}{\partial \alpha_i} \quad (22)$$

$$\alpha_i^n = \alpha_i^{n-1} - \delta \alpha_i^n \quad (23)$$

where,  $\mu$  represents momentum and  $\epsilon$  learning rate. Since, there is no constraint on the sign of  $\alpha_i$ , PReLU does not necessarily need to be monotonic.

#### 2.5.4 ELU (Exponential Linear Unit)

ELU was proposed by Clevert et al., [27] as an improvement to ReLU activation function and its variants (LReLU and PReLU). ELU activation function is defined as



**Fig. 8** **a** The exponential linear unit (ELU) with  $\alpha = 1$  and its corresponding gradient. **b** The scaled exponential linear unit (SELU) with  $\alpha = 1.6733$  and  $\lambda = 1.0507$ , with its corresponding gradient

$$elu(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \quad (24)$$

where,  $\alpha$  is a hyper parameter that controls the value for negative inputs. The gradient of ELU is given by

$$\frac{dalu(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ elu(x) + \alpha & \text{if } x \leq 0 \end{cases} \quad (25)$$

ELU also solves the problem of vanishing gradient similar to ReLU by having unit gradient for all positive inputs. Whereas, its gradient is non-zero for negative inputs pushing the mean of the activation function closer to zero. As discussed in Proposition 1, this leads to faster training of the neural network. Major improvement of ELU over LReLU and PReLU comes from its saturating behavior for negative inputs. This brings relatively less variation in activation function which in turn makes its noise robust. Moreover, the ELU activation has shown improved results over ReLU activation on both supervised and unsupervised machine learning tasks. Figure 8a shows ELU activation function and its gradient for hyper parameter  $\alpha = 1$ .

### 2.5.5 SELU (Scaled Exponential Linear Unit)

It is well known that, even though FNNs (Fully Connected Neural Networks) are highly sophisticated machine learning algorithm, they fail to stand up to their reputation in real life. However, with the support of CNNs and RNNs, NNs can achieve state of the art results. This can be attributed to CNNs and RNNs having parameter sharing across feature maps and cells respectively combined with normalization techniques like batch normalization and layer normalization. Since both image and time series data have structure across space and time respectively, these parameters are efficiently shared, and exponentially reduce the complexity. In [28], authors find

that the reason behind lacking performance of FNNs is their high variance across different training examples and sensitivity to perturbations. This brings us to the concept of SNNs. SNNs (Self Normalized Neural Networks) keep normalization of activation function when propagating them through layers of networks. SNNs need two things to work, first is a custom weight initialization and SELUs as activation function. The weight initialization allows the mean and variance at each layer to be zero and one respectively.

SNNs can not be derived with ReLUs, sigmoid, tanh and leaky ReLU. SELUs are obtained by multiplying the exponential linear unit with a parameter  $\lambda$  which is kept greater than 1 to ensure slope greater than one. The authors of SELU provide four conditions which any activation function should ideally possess (which SELUs do follow) are:

1. Both positive and negative values should be present in the range of the activation function so that the mean is zero.
2. Saturating regime in the activation function to dampen or reduce the variance of the output of activation.
3. A regime with slope greater than one so as to increase the variance if needed by the network.
4. The activation function should be continuous.

SELU (Scaled Exponential Linear Unit) is mathematically defined as

$$selu(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda\alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \quad (26)$$

where,  $\alpha$  and  $\lambda$  are two fixed parameters derived from the input. For standard scaled inputs authors provide optimal value of  $\alpha$  as 1.6733 and  $\lambda$  as 1.0507. The gradient of SELU can be written as

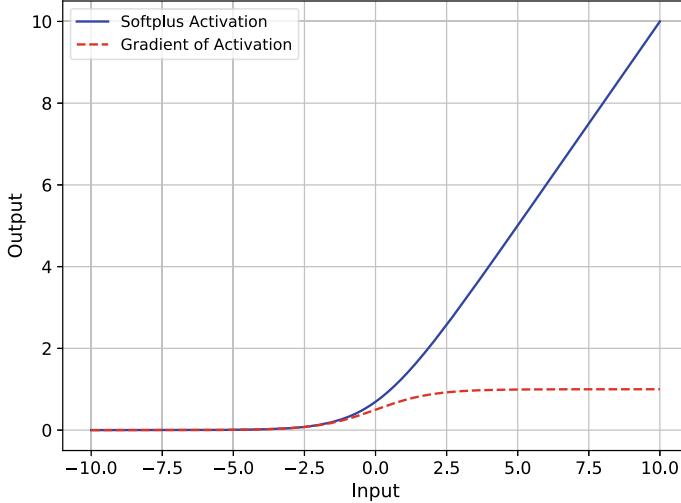
$$\frac{dselu(x)}{dx} = \begin{cases} \lambda & \text{if } x > 0 \\ \lambda\alpha + selu(x) & \text{if } x \leq 0 \end{cases} \quad (27)$$

The characteristics of SELU are shown in Fig. 8b. SELU has self normalizing property that allows to train networks with high learning rate that have many layers. This activation function has no exploding and vanishing gradient problems.

## 2.6 Softplus

Softplus is another activation proposed in [24]. It is a smooth approximation of ReLU which is  $\max(0, x)$ . It is mathematically defined as:

$$Softplus(x) = \log(1 + e^x) \quad (28)$$



**Fig. 9** Softplus activation function and its corresponding gradient

Gradient of softmax can be written as a sigmoid unit:

$$\frac{d(\text{Softplus}(x))}{dx} = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} = \sigma(x) \quad (29)$$

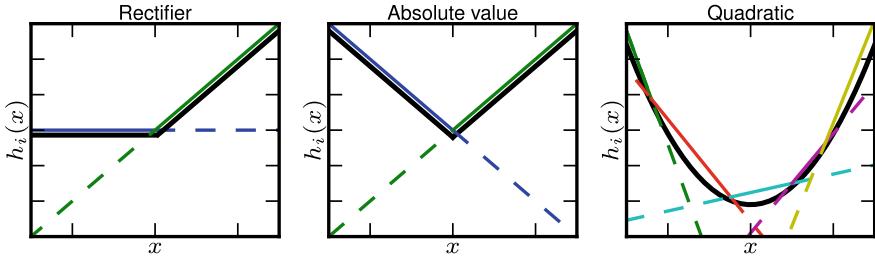
Softplus activation function is computationally complex but provides many advantages over other activations. Since the derivative of a softplus unit is a sigmoid, it does not suffer with the vanishing gradient issue. The gradient does not die in almost half of the whole real domain. Unlike ReLU, the gradient does not become zero instantly when the input is negative. So, the neural network does not have many dead neurons. As mentioned in ReLU, it is questionable to quantify the effect sparsity in neural networks. In the next subsection, we will discuss maxout networks, that have no sparsity in activation, however, they perform better than ReLU activation (Fig. 9).

## 2.7 Maxout

Maxout was proposed by Goodfellow [29], as a completely different class of activation function. We first see the mathematical description of maxout and then its specific benefits. We consider a hidden layer with  $t$  neurons, where each neuron is denoted as  $h_i$  ( $i \in \{1, \dots, t\}$ ). The neuron  $h_i$  after applying maxout can be written as:

$$h_i = \max_{j \in 1, \dots, k} z_{ij} \quad (30)$$

$$z_{ij} = x^T W_{...ij} + b_{ij} \quad (31)$$



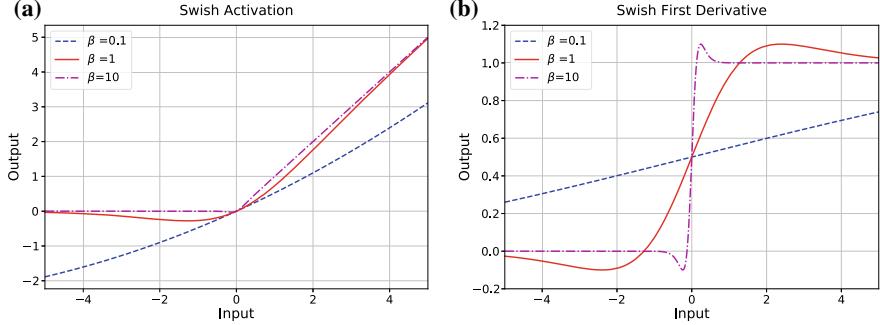
**Fig. 10** Image Courtesy: [29]. How maxout can approximate arbitrary uni-variate functions. Similarly, multivariate functions can also be approximated by maxout

where, input  $x \in \mathbb{R}^d$ ,  $W$  is the weight tensor of shape  $(d \times t \times k)$  and  $b$  is a matrix of shape  $(t \times k)$ . In case of convolutional neural networks, the maxout unit is implemented across  $k$  different feature maps. It will amount to maxpooling operation across spacial domain and maxout across channels. Reference [29] mentions that similar to multilayer perceptron network, a maxout network is also an universal function approximator. They show that, given enough affine components ( $t$  of a layer),  $k = 2$  is sufficient to approximate any arbitrary continuous function. They use the fact that any continuous function can be approximated (with error  $\epsilon$ ) using piece wise linear function. It is important to note that maxout will learn a piece wise linear function, which will serve as an activation function to the neural network as shown in Fig. 10.

Maxout networks are also advantageous because of their compatibility with dropout regularization technique. Dropout is basically nullifying the neurons of a network, randomly. It is implemented by masking the hidden layer with a same dimensional tensor of zeros and ones. Dropout results in an ensemble of multiple weaker models providing regularization effect. The output of a maxout, when trained with dropout, changes relatively rarely with different masks of dropout. Maxout networks basically enlarge their linear regions, so that even if some units are dropped, the input to maxout still is such that it falls in the same linear region. This invariance is easily achieved with maxout networks making it an excellent activation function. One of the disadvantage of maxout networks is the need of larger architecture sizes to allow for implementation of maxout operation. The extra dimension  $k$ , increases the parameters of the model to some extent. However, their performance is remarkable considering such a non-trivial form of non linearity when compared with ReLUs.

## 2.8 Swish Activation Function

So far, we have seen various activations that have been mostly hand-designed to improve certain characteristics of ReLU activation function. In [30], authors present a method to discover novel activation functions using automated reinforcement



**Fig. 11** **a** The Swish activation function. **b** First derivative of swish

learning based search. The core idea is to design a composite combination of various existing activation functions and empirically compare them to find the best one. In other words, design a search space consisting of composite combination of existing functions and then test each composite combination on a standard data set to compare them with each other. The crux of the method lies in designing the composite combination of existing functions. Using an exhaustive search, i.e., trying all combinations of activation functions would result in a very large search space making the search practically infeasible. Hence, in [30], authors used a RNN controller to predict different components of an activation function and feed it back to predict the other components of the same new activation function.

Now, once an activation function has been found, it is tested on CIFAR-10 data set using a child network. A list of top performing functions is maintained to keep track of best performing activations. This method resulted in various novel activation functions which outperformed ReLU on CIFAR-10 data set (explained in 3) atleast using the child network. It was found that the activation function  $f(x) = x \cdot \sigma(\beta x)$  outperformed ReLU on both CIFAR-10 and CIFAR-100 on various Deep Architectures. The function  $f(x) = x \cdot \sigma(\beta x)$  is called **Swish Activation function**. Here,  $\beta$  is a constant or trainable parameter and  $\sigma(z) = (1 + \exp(-z))^{-1}$ . Figure 11 shows swish activation function and its first derivate for various values of  $\beta$ . It can be seen that for  $\beta = 0$ ,  $f(x) = \frac{x}{2}$  and behaves similar to identity function. As  $\beta \rightarrow \infty$ ,  $f(x) \rightarrow \max(0, x)$  or  $f(x)$  acts as ReLU activation function. This suggests that the swish activation function can be loosely viewed as a smooth version of the ReLU activation function.

### 3 Comparison of Activation Functions

After understanding various existing activation functions, it is important to see how these activation functions perform with different architectures. To compare them, we present the results of these activation functions on CIFAR-10 and CIFAR-100 data

sets using 3 different state of the art architectures. CIFAR-10 and CIFAR-100 data sets contain 60,000 colored images belonging to 10 and 100 classes respectively. 50,000 of these images belong to training set and the rest 10,000 are the test images. The task is to accurately classify the test images based on training data set. The 3 different architectures used are ResNet-164 [31], Wide ResNet 28-10 (WRN) [32] and DenseNet 100-12 [33]. The results shown are taken from [30].

Tables 1 and 2 showcase the test accuracy of several non linear activation functions on three different architectures. It is evident that no single activation works best on every architecture. For example, in case of CIFAR-10 data set, Softplus outperforms all other activation functions when ResNet is used as the underlying architecture but performs poorly on Wide ResNet as compared to other activations. This creates a dilemma around how to select the optimal activation function for any architecture. Hence, we want an activation function which can adapt itself depending on the data set and architecture. In the next section, we will focus on such activation functions and discuss two different approaches that can be used to learn activation functions.

**Table 1** CIFAR-10 test accuracy

Model	ResNet	WRN	DenseNet
LReLU	94.2	95.6	94.7
PReLU	94.1	95.1	94.5
Softplus	94.6	94.9	94.7
ELU	94.1	94.1	94.4
SELU	93.0	93.2	93.9
ReLU	93.8	95.3	94.8
Swish	94.5	95.5	94.8

**Table 2** CIFAR-100 test accuracy

Model	ResNet	WRN	DenseNet
LReLU	74.2	78.0	83.3
PReLU	74.5	77.3	81.5
Softplus	76.0	78.4	83.7
ELU	75.0	76.0	80.6
SELU	73.2	74.3	80.8
ReLU	74.2	77.8	83.7
Swish	75.1	78.0	83.9

## 4 Learning Activation Functions

Till now, we have discussed various activation functions starting from Identity function to Swish. In this section, we shift our focus from fixed activation functions to those which can be learned. Learning the activation functions means the shape of the function is not fixed but is parameterized by learnable weights which are learned during training of NN. Owing to their capability of adaptation, they are also known as ‘Adaptive Activation Functions (AAFs)’. References [34–37] show different approaches to design AAFs. Below, we discuss two separate approaches to learn activation functions. The first method uses an ‘Adaptive Piecewise Linear Activation Function’ [37] which is learned independently for each neuron using gradient descent. Next, we propose a unique technique which aims to learn an activation function using techniques of non linear approximation. We call it ‘Self Learnable Activation Function (SLAF)’.

### 4.1 Learning Adaptive Piecewise Linear (APL) Activation Functions

As the name suggests, this method learns a piecewise linear activation function for each neuron in the neural network. It formulates an activation function  $h(x)$  as,

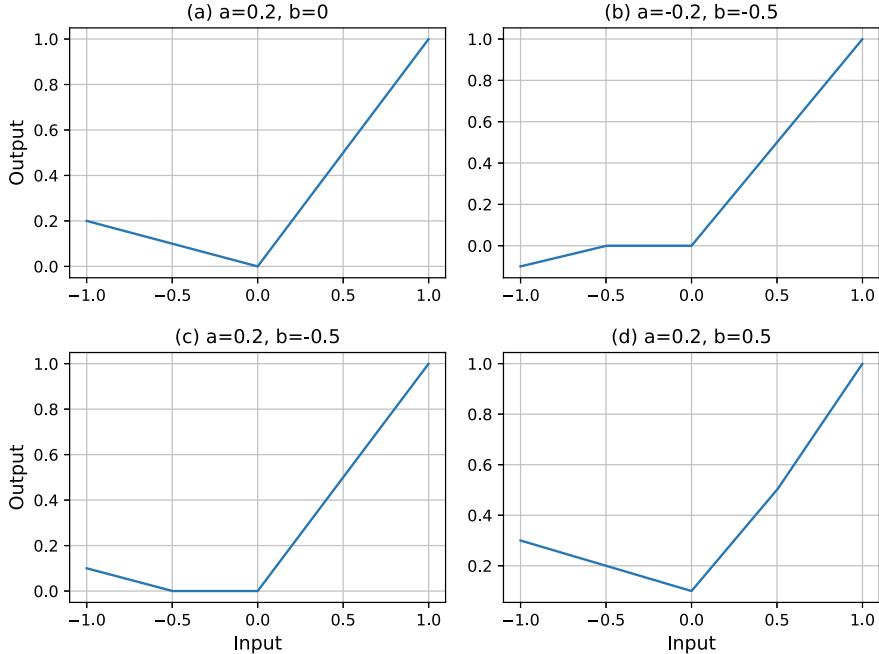
$$h(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s) \quad (32)$$

where,  $S$  is a hyper parameter and the variables  $a_i^s$  and  $b_i^s$  for  $i \in 1, \dots, S$  are learned using the same algorithm as the other network weights are learned. Note that the method aims to learn the best piecewise linear function for a given data set and architecture. Hence, Eq. (32) should span the entire space of continuous piece wise linear functions.

**Theorem 1** Any continuous piecewise linear function  $g(x)$  can be expressed by Eq. (32) for some  $S$  if it satisfies the following two conditions:

1. There exists a scalar  $u$  such that  $g(x) = x$  for  $x \geq u$
2. There exists two scalars  $\alpha$  and  $v$  such that  $\nabla_x g(x) = \alpha$  for all  $x < v$

We are not providing the proof of above theorem but reader may refer to [37] for further details on above result. Figure 12 shows APL activation function when summation in Eq. (32) has only 1 term. It is very interesting to note all the curves except (b) show non-monotonic behavior of activation function which is contrary to the behavior observed in fixed activation function (other than swish activation function). Moreover, Fig. 12b shows the non-convex behavior of the activation function. This show the freedom of APL activation functions to adapt themselves depending on the task. Above activation function also outperforms ReLU on various data sets. As



**Fig. 12** Adaptive piece wise linear activation function with different  $a, b$  parameters for  $S = 1$

mentioned in [37], the best error rate on CIFAR-10 using APL activation function is 7.51% whereas ReLU had an error rate of 7.73% using Network in Network (NIN) architecture [38]. Similarly, on the same architecture for CIFAR-100 data set, APL outperforms ReLU by around 2% by achieving an error rate of 30.83%.

Above method enables neural networks to learn diverse set of activation functions. All these learned activation functions will be piecewise linear. Hence, the search space explored by this method is limited to only piecewise linear functions. Below, we discuss a more generalized method of learning activation functions which doesn't make any inherent assumptions about the nature of activation functions.

## 4.2 SLAF: Learning Non-linear Approximation of Activation Functions

Every continuous function in a function space can be written as a linear combination of basis functions in the same function space [39]. If  $\phi_i$ 's are the basis elements, then the function  $f(x)$  with input  $x$  can be expressed as,

$$f(x) = \sum_{i=0}^{i=\infty} a_i \phi_i(x) \quad (33)$$

Here,  $a_i$ s are the coefficients of basis elements and are unique to  $f(x)$ . If we fix the basis function and learn their coefficients using a suitable algorithm, we can effectively learn  $f(x)$ . The only problem here is that the expression contains infinite elements and it is practically impossible to learn all of them. Restricting the number of elements in the basis results in an approximation of the function. A suitable approximation of a function  $f(x)$  with  $N$  basis elements can be given by:

$$f(x) \approx a_0 \phi_0(x) + a_1 \phi_1(x) + \dots + a_{N-1} \phi_{N-1}(x) = \tilde{f}(x) \quad (34)$$

where,  $\phi_i$  are the basis elements and  $a_i$  are the corresponding coefficients. If we take  $f(x)$  to be the activation function which we aim to learn, learning  $\{a_0, \dots, a_{N-1}\}$  would eventually learn  $\tilde{f}(x)$ . Since, this method intends to learn the approximation  $\tilde{f}(x)$  and not the actual function  $f(x)$ , it becomes a prime concern to find a basis which provides a good approximation with  $N$  basis elements. Although there can be many choices for basis functions, we use Even Mirror Fourier Non-linear (EMFN) Filter Basis owing to its strong approximation capabilities.

#### 4.2.1 Even Mirror Fourier Non-linear (EMFN) Filter

EMFN filters [40] can be used for approximation of any continuous function  $f(x)$  in the interval  $[-1, 1]$ . So, an extension of  $f(x)$  in the EMFN basis on entire real axis  $\mathbb{R}$  is considered by taking its periodic even mirror repetition. To do this, the values of  $f(x)$  lying between  $[-1, 1]$  are taken and repeated on entire real line. The repetition is done to satisfy the following two properties,

$$\begin{aligned} f(1+x) &= f(1-x) : \text{Even Mirror of } f(x) \\ f(x) &= f(x+4) : \text{Periodic extension} \end{aligned} \quad (35)$$

The EMFN filters use sine and cosine functions as basis elements. Since,  $f(x)$  is periodic with period 4, it is easy to write the Fourier series expansion of  $f(x)$ . The Fourier series expansion contains the following basis elements,

$$\begin{aligned} &\{1, \cos\left(\frac{\pi}{2}x\right), \sin\left(\frac{\pi}{2}x\right), \cos(\pi x), \sin(\pi x), \cos\left(\frac{3\pi}{2}x\right), \\ &\quad \sin\left(\frac{3\pi}{2}x\right), \cos(2\pi x), \sin(2\pi x), \cos\left(\frac{5\pi}{2}x\right), \dots\} \end{aligned} \quad (36)$$

Now, since the basis elements  $\{\cos\left(\frac{\pi}{2}x\right), \sin(\pi x), \cos\left(\frac{3\pi}{2}x\right), \sin(2\pi x)\}$  don't satisfy even mirror property of  $f(x)$ , they can be removed from the basis function.

Hence, the resultant basis and the corresponding function approximation can be given as:

$$\{1, \sin\left(\frac{\pi}{2}x\right), \cos(\pi x), \sin\left(\frac{3\pi}{2}x\right), \cos(2\pi x), \dots\} \quad (37)$$

$$f(x) \approx a_0 + a_1 \sin\left(\frac{\pi}{2}x\right) + a_2 \cos(\pi x) + a_3 \sin\left(\frac{3\pi}{2}x\right) + \dots \quad (38)$$

It is also possible to find an extension of above basis function for approximation of function in  $N$  dimensions but we will restrict our discussion to 1-dimension due to the lack of relevance.

#### 4.2.2 Model Setup

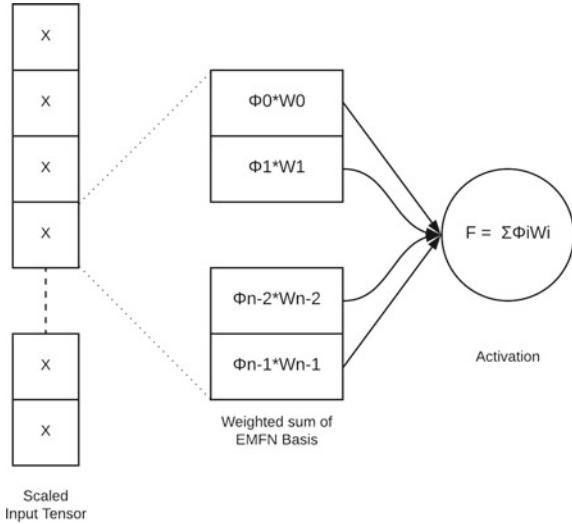
We want to use EMFN Filters approximation method to learn activation functions which can adapt itself depending on the task and architecture. First, we replace the existing pre-defined “Activation-Function” block by ‘**Self-Learnable Activation-Function (SLAF)**’ block. This ‘Self-Learnable Activation-Function block’ should eventually learn a good approximation of the optimal activation function for any given network. Let the optimal activation function for the network be  $F$ . An approximation of  $F$  can be given by Eq. (38). The coefficients in Eq. (38) can theoretically represent  $F$  with arbitrarily small error. Moreover, for a fixed input and fixed basis function (EMFN basis), the only variable in Eq. (38) are the coefficients. Hence, if we learn these coefficients, we would eventually learn the optimal activation function. So, we can now narrow our goal of learning activation functions to simply finding a model where we can plug the approximation equation in place of activation function and learn the right set of coefficients along with other network weights.

Let  $x$  be the input to the activation function and  $f(x)$  denote its output. We know that domain of approximation of  $f(x)$  using EMFN filters is  $[-1, 1]$ . In general, the input to an activation function in a neural network can take any real value in the range  $(-\infty, \infty)$ . Hence, before using the EMFN filter approximation, we first need to restrict the input  $x$  to the range  $[-1, 1]$ . This can be done by dividing the input tensor  $X$  ( $X$  is the tensor representation of  $x$ , containing  $B$   $x$ 's, where  $B$  is the batch size) by its absolute maximum value (across batch for each feature). This would bound the range of transformed tensor between  $[-1, 1]$ . We also divide this transformed tensor by a learnable parameter  $m$  to further narrow its range between  $[-1, 1]$ . This is done to avoid abrupt behavior of EMFN filters around  $x = \{-1, 1\}$ . This gives us the transformed tensor  $\hat{X}_1$ ,

$$\hat{X}_1 = \frac{X}{m \cdot \max(|X|)} \quad (39)$$

It is possible that during training phase,  $\max(|X|)$  might be 0 or a very small positive quantity. This would lead to division by 0 in Eq. (39). Hence, we add a small learnable

**Fig. 13** SLAF model using EMFN filter



parameter  $\epsilon$  in the denominator of Eq. (39). This gives us the final transformed tensor  $\hat{X}$  which is defined as follows:

$$\hat{X} = \frac{X}{m \cdot \max(|X|) + \epsilon} \quad (40)$$

We keep both  $m$  and  $\epsilon$  as learnable parameters as they both are data set dependent. Now, we can use this scaled input tensor for approximation of our activation function. Using this, we can define our activation function as,

$$\begin{aligned} f(x) &= f_{SLAF}(\hat{x}) \\ f_{SLAF}(\hat{x}) &= W_0\phi_0(\hat{x}) + \dots + W_{n-1}\phi_{n-1}(\hat{x}) \end{aligned} \quad (41)$$

where  $W_i$ 's are learnable and  $\phi_i$ 's belong to the EMFN basis and  $\hat{x}$  is one element of  $\hat{X}$  (which has the same shape as  $X$ ). Note that,  $W_i$ 's will be shared across the complete tensor  $X$  to avoid over-fitting. This is pictorially depicted in Fig. 13.

#### 4.2.3 Training Routines

EMFN filters have strong expressive power which means that the coefficients of basis elements learned by model can highly over fit to training data set resulting in poor generalization. To avoid this, we need improved training routines. We propose following methods which can be used along with SLAF to improve the generalization.

1. **Regularization** is a standard technique to reduce over fitting in machine learning. L2 regularization [9] is used on the coefficients being learned. Reference [41]

states the problem in using L2 regularization with Adam optimizer [42]. Hence, if Adam optimizer is being used for optimizing network weights, a separate optimizer is used for regularization loss. The experiments in below sections have used SGD optimizer for the regularization loss and adam optimizer for minimizing the cross entropy or mean squared loss depending the task.

2. **Learning rate decay:** Learning rate decay [43, 44] is essentially very important for most of the tasks when using the self-learnable activation function in the neural network. This helps in avoiding local minima.
3. **Tuning the number of basis elements:** The number of basis elements change the expressive power of the network. High number of basis elements can not only lead to over fitting but also raise convergence issues. The experiments in the below subsection have used only 3 or 4 basis elements.

#### 4.2.4 Experiments

In this subsection, we present series of experiments where the results of fixed activation function and self-learnable activation function are compared for different tasks.

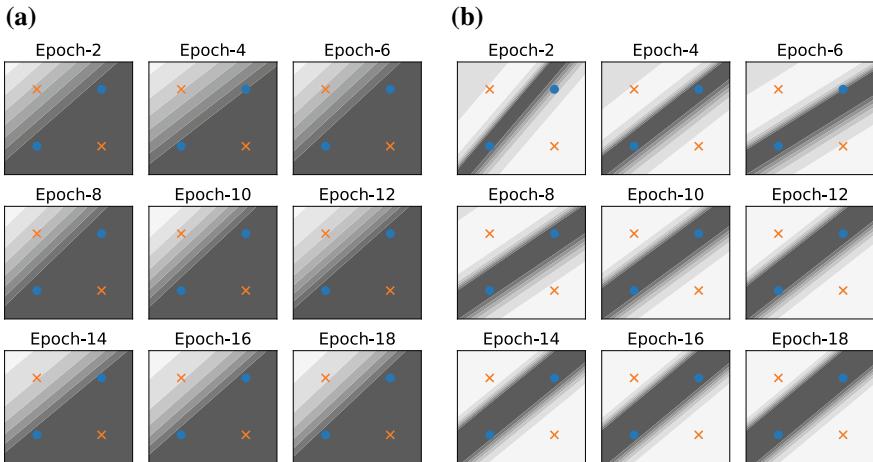
1. **XOR:** XOR is a logical operator which takes its two binary inputs and its output is also a binary value. Table 3 shows this classification (“x” and “o” denote two separate classes). It is clear that XOR operation is not linearly separable and therefore, it is impossible to learn it without using hidden layer or simply with a “Perceptron”. The architecture used for this experiment first takes a weighted combination of inputs and then apply an activation function (acting as a non-linearity) on the output of this weighted combination. The main reason for using this sort of architecture is to see whether the existing activation function have enough capacity to learn this decision boundary or not. Table 4 shows that the maximum accuracy that can be achieved using ReLU is 75% whereas SLAF can classify this data set with 100% accuracy. This is because SLAF can adapt to the task depending on the data set. The final decision boundaries learned by both the activation functions are shown in Fig. 14.
2. **MNIST:** MNIST data set contains 70,000 images of  $28 \times 28$  pixels each containing a hand written digit from 0 to 9. The task is to classify these images into 10 classes depending on the letter written in image. We use Convolutional Neural Network (CNN) [45] consisting of 2 convolutional layers followed by 2 fully connected layers to train our model. The architecture uses 3 activation functions. We replace all three activation functions by SLAF. We used L2 regularization on SLAF weights and learning rate decay, and achieve an accuracy of 99.46%. The

**Table 3** XOR operator

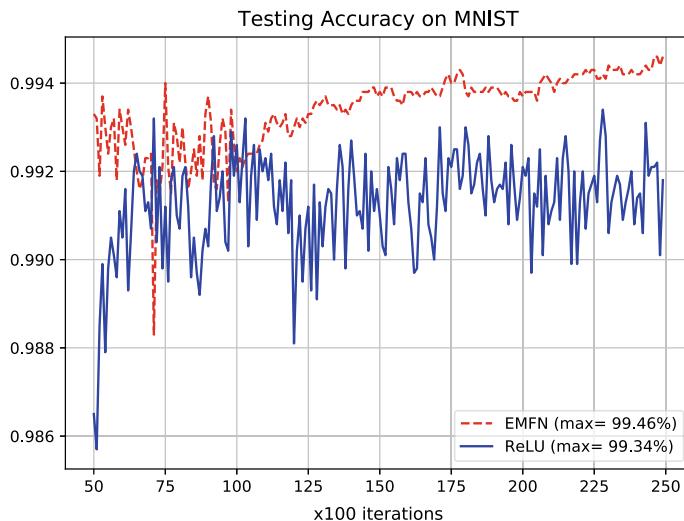
	0	1
0	x	o
1	o	x

**Table 4** Results on XOR problem.  $k_1, k_2$  are the number of elements used for sine and cosine terms respectively

Activation function	Accuracy
ReLU	0.75
EMFN filter $k_1 = 3$ $k_2 = 3$	1.0



**Fig. 14** Comparison of decision boundaries learned with training epochs by using two different activation functions. ‘o’ (dot) refers to the class labeled as zero and ‘x’ (cross) refers to the class labelled as one. **a** Using ReLU **b** Using SLAF



**Fig. 15** Test accuracy versus iterations on MNIST data set using different activation functions

maximum accuracy achieved using ReLU on the same architecture was 99.34%. Figure 15 shows that the testing accuracy using SLAF is almost always better than ReLU. Moreover, the accuracy curve of neural network using SLAF had negligible fluctuation curve as compared to ReLU showing the much stronger generalization capability of SLAF.

## 5 Conclusion

In this chapter, we first understood neural networks were biologically motivated and then understood the role of activation functions in neural networks. Activation functions are non linear mappings from one subspace to another, allowing neural networks to learn arbitrarily complex functions which forms the basis of deep learning. In the second part, we studied a detailed description of all popular activation functions that have been proposed so far, and saw how this area has been developing with rather less mathematical evidences. The nature of these functions heavily affect the training dynamics which is why researchers have been focusing on designing better activations for the past decade. Concepts such as normalization of activations, exploding and vanishing gradients have served as the key to these feats in the synthesis of newer activations. We also learnt that different activation functions lead to different performances of neural networks. In the later parts of the chapter, the paradigm of learnable activation functions was introduced. We discussed the adaptive piece wise activation and a novel self learnable activation function. Both of the approaches stressed on developing a way to learn the activation functions while training the neural network depending on the kind of data set, task and the network. Since it is well known that a high variation in performance is experienced while switching between different activation functions, it becomes incorrect to claim that a fixed activation in all circumstances will outperform any other activation on every task/dataset. Hence, an algorithm is proposed which can search for the best activation in the entire space of continuous functions. The readers should take a note that the algorithm proposed in this chapter, i.e., self learnable activation functions uses even mirror fourier nonlinear filter basis in its current form but it may not be the perfect way to learn the best activation. However, it indeed points towards a promising direction which might completely change our perception of neural networks. There are yet many unknowns which act as barriers for complete understanding of neural networks. Activation function yet remains to be the biggest black box which has been driven through isolated concepts, or biological analogies. For faster developments in this field people look through the lens of better empirical performance, but an approach for learning activations was introduced in this work with the aim of creating a mathematical basis behind these non linearities.

## 6 Future Works

The chapter highlights the importance of learnable activation functions and discusses two completely different methods to learn them. The performance of SLAF is shown over simple data sets. To empirically validate the usefulness of SLAF, one must conduct experiments on more complex data sets such as CIFAR-100 and Image Net. The basic methodology used by EMFN filters to learn non-linear approximation of activation functions is discussed in Sect. 4.2. Different basis functions can be used in place of EMFN filters to empirically find the most suitable one. A proper model setup must be designed for every basis. More training routines, such as, applying dropout and different regularization techniques on the activation coefficients can be proposed to achieve faster optimization of the neural networks. The chapter focuses on only one block of DNNs, viz. activation functions. One can always extend the concept of learning to other non-linear components present in the neural networks.

## References

1. Bengio, Y., Aaron, C., Courville, Vincent, P.: Unsupervised feature learning and deep learning: a review and new perspectives. In: CoRR abs/1206.5538 (2012). [arXiv:1206.5538](https://arxiv.org/abs/1206.5538). <http://arxiv.org/abs/1206.5538>
2. Zhang, G.P.: Neural networks for classification: a survey . IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **30**(4), 451–462 (2000). ISSN 1094-6977. <https://doi.org/10.1109/5326.897072>
3. Tian, G.P., Pan, L.: Predicting short-term traffic flow by long short-term memory recurrent neural network. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), pp. 153–158 (2015). <https://doi.org/10.1109/SmartCity.2015.63>
4. Wiki. Activation Potential | Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/wiki/Action\\_potential](https://en.wikipedia.org/wiki/Action_potential) (2018). Accessed 31 Dec 2018
5. Stanford CS231n—Convolutional neural networks for visual recognition. <http://cs231n.github.io/neural-networks-1/>. Accessed 01 May 2019
6. London, M., Häusser, M.: Dendritic computation. Annu. Rev. Neurosci. **28**(1), 503–532 (2005). <https://doi.org/10.1146/annurev.neuro.28.061604.135703>
7. Wiki. Activation Function | Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function) (2018). Accessed 31 Dec 2018
8. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Statist. **22**(3), 400–407 (1951). <https://doi.org/10.1214/aoms/1177729586>
9. Krogh, A., Hertz, J.A.: A simple weight decay can improve generalization. In: Proceedings of the 4th International Conference on Neural Information Processing Systems, NIPS’91, pp. 950–957. Morgan Kaufmann Publishers Inc., Denver, Colorado. <http://dl.acm.org/citation.cfm?id=2986916.2987033> (1991). ISBN 1-55860-222-4
10. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. J. R. Stat. Soc. Ser. B **67**, 301–320 (2005)
11. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: CoRR abs/1502.03167 (2015). [arXiv:1502.03167](https://arxiv.org/abs/1502.03167). <http://arxiv.org/abs/1502.03167>
12. Autoencoders. <https://en.wikipedia.org/wiki/Autoencoder>. Accessed 05 Sept 2019

13. Saxe, A.M., McClelland, J., Ganguli, G.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks (2013)
14. Arora, S., et al.: A convergence analysis of gradient descent for deep linear neural networks. In: CoRR abs/1810.02281 (2018). [arXiv:1810.02281](https://arxiv.org/abs/1810.02281). <http://arxiv.org/abs/1810.02281>
15. Toms, D.J.: Training binary node feedforward neural networks by back propagation of error. Electron. Lett. **26**(21), 1745–1746 (1990)
16. Muselli, M.: On sequential construction of binary neural networks. IEEE Trans. Neural Netw. **6**(3), 678–690 (1995)
17. Ito, Y.: Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Netw. **4**(3), 385–394 (1991)
18. Kwan, H.K.: Simple sigmoid-like activation function suitable for digital hardware implementation. Electron. Lett. **28**(15), 1379–1380 (1992)
19. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics. Springer, Berlin (2006). ISBN 0387310738
20. Parkes, E.J., Duffy, B.R.: An automated tanh-function method for finding solitary wave solutions to non-linear evolution equations. Comput. Phys. Commun. **98**(3), 288–300 (1996)
21. LeCun, Y., et al.: Efficient backprop In: Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop, pp. 9–50. Springer, Berlin (1998). ISBN: 3-540-65311-2. <http://dl.acm.org/citation.cfm?id=645754.668382>
22. Pascanu, R., Mikolov, R., Bengio, Y.: Understanding the exploding gradient problem. In: CoRR abs/1211.5063 (2012). [arXiv:1211.5063](https://arxiv.org/abs/1211.5063). <http://arxiv.org/abs/1211.5063>
23. Hahnloser, R.H.R., et al.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature **405**, 947 (2000). <https://doi.org/10.1038/35016072>
24. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10, pp. 807–814. Omnipress, Haifa, Israel (2010). ISBN 978-1-60558-907-7. <http://dl.acm.org/citation.cfm?id=3104322.3104425>
25. Maas, A.L.: Rectifier nonlinearities improve neural network acoustic models. In: ICML, vol. 30 (2013)
26. He, K., et al.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034 (2015)
27. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). In: arXiv preprint (2015). [arXiv:1511.07289](https://arxiv.org/abs/1511.07289)
28. Klambauer, G., et al.: Self-normalizing neural networks. In: Advances in Neural Information Processing Systems, pp. 971–980 (2017)
29. Goodfellow, I., et al.: Maxout networks. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning, vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1319–1327. <http://proceedings.mlr.press/v28/goodfellow13.html>
30. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions (2018)
31. He, K., et al.: Identity mappings in Deep residual networks. In: CoRR abs/1603.05027 (2016). [arXiv:1603.05027](https://arxiv.org/abs/1603.05027). <http://arxiv.org/abs/1603.05027>
32. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: CoRR abs/1605.07146 (2016). [arXiv:1605.07146](https://arxiv.org/abs/1605.07146). <http://arxiv.org/abs/1605.07146>
33. Huang, G., et al.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269 (2017). <https://doi.org/10.1109/CVPR.2017.243>.
34. Yu, C.C., Tang, Y.C., Liu, B.D.: An adaptive activation function for multilayer feedforward neural networks. In: 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM '02. Proceedings, vol. 1, pp. 645–650 (2002). <https://doi.org/10.1109/TENCON.2002.1181357>.
35. Qian, S., et al.: Adaptive activation functions in convolutional neural networks. Neurocomputing **272**, 204–212 (2018). ISSN 0925-2312. <https://doi.org/10.1016/j.neucom.2017.06.070>.

36. Hou, L., et al.: ConvNets with smooth adaptive activation functions for regression. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. vol. 54. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, pp. 430–439 (2017). <http://proceedings.mlr.press/v54/hou17a.html>
37. Agostinelli, F., et al.: Learning activation functions to improve deep neural networks. In: CoRR abs/1412.6830 (2014). [arXiv:1412.6830](https://arxiv.org/abs/1412.6830). <http://arxiv.org/abs/1412.6830>
38. Lin, M., Chen Q., Yan, S.: Network in network. In: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, 14–16, 2014 Conference Track Proceedings, (2014). <http://arxiv.org/abs/1312.4400>
39. Basis function. June 2018. [https://en.wikipedia.org/wiki/Basis\\_function](https://en.wikipedia.org/wiki/Basis_function)
40. Carini, A., Sicuranza, G.L.: Even mirror Fourier nonlinear filters. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5608–5612 (2013)
41. Loshchilov, L., Hutter, F.: Fixing weight decay regularization in adam. In: CoRR abs/1711.05101 (2017). [arXiv:1711.05101](https://arxiv.org/abs/1711.05101). <http://arxiv.org/abs/1711.05101>
42. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015, Conference Track Proceedings, (2015). <http://arxiv.org/abs/1412.6980>
43. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011). ISSN 1532-4435. <http://dl.acm.org/citation.cfm?id=1953048.2021068>
44. Zeiler, M.D.: ADADELTA: An adaptive learning rate method. In: CoRR abs/1212.5701 (2012). [arXiv:1212.5701](https://arxiv.org/abs/1212.5701). <http://arxiv.org/abs/1212.5701>
45. Krizhevsky, A., Sutskever, I., Hinton, G.E.: imagenet classification with deep convolutional neural networks. In: Pereira, F., et al. (eds.) Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105 (2012). <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

# Adversarial Examples in Deep Neural Networks: An Overview



Emilio Rafael Balda, Arash Behboodi and Rudolf Mathar

**Abstract** Deep learning architectures are vulnerable to adversarial perturbations. They are added to the input and alter drastically the output of deep networks. These instances are called adversarial examples. They are observed in various learning tasks from supervised learning to unsupervised and reinforcement learning. In this chapter, we review some of the most important highlights in theory and practice of adversarial examples. The focus is on designing adversarial attacks, theoretical investigation into the nature of adversarial examples, and establishing defenses against adversarial attacks. A common thread in the design of adversarial attacks is the perturbation analysis of learning algorithms. Many existing algorithms rely implicitly on perturbation analysis for generating adversarial examples. The summary of most powerful attacks are presented in this light. We overview various theories behind the existence of adversarial examples as well as theories that consider the relation between the generalization error and adversarial robustness. Finally, various defenses against adversarial examples are also discussed.

**Keywords** Adversarial examples · Deep learning · Classification · Regression · Perturbation analysis · Statistical learning · Adversarial training · Adversarial defenses

---

E. R. Balda · A. Behboodi (✉) · R. Mathar

Institute for Theoretical Information Technology (TI), RWTH Aachen University,  
ICT cubes, Kopernikusstraße 16, 52074 Aachen, Germany  
e-mail: [behboodi@ti.rwth-aachen.de](mailto:behboodi@ti.rwth-aachen.de)

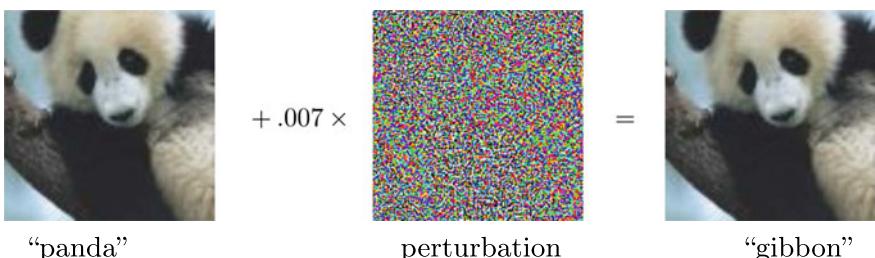
E. R. Balda  
e-mail: [balda@ti.rwth-aachen.de](mailto:balda@ti.rwth-aachen.de)

R. Mathar  
e-mail: [mathar@ti.rwth-aachen.de](mailto:mathar@ti.rwth-aachen.de)

## 1 Introduction

Artificial intelligence is on the rise and Deep Neural Networks (DNNs) are an important part of it. Whether it is in speech analysis [29] or visual tasks [26, 34, 58, 68], they shine with a performance beyond what was imagined a decade ago. Their success is undeniable, nevertheless a flaw has been spotted in their performance. They are not stable under adversarial perturbations [69]. Adversarial perturbations are intentionally worst case designed noises that aim at changing the output of a DNN to an incorrect one. The perturbations are most of the time so small that an ordinary observer may not even notice it, and even the state-of-the-art DNNs are highly confident in their, wrong, classification of these adversarial examples. This phenomena is depicted in Fig. 1, borrowed from [24], where a subtle adversarial perturbation is able to change the classification outcome. Robustness to adversarial perturbations is different from robustness to random noise [19], a trait that can be achieved by DNNs. The existence of adversarial perturbations was known for machine learning algorithms [9], however, they were first noticed in deep learning research in [69]. These discoveries generated interest among researchers to understand the instability of DNNs, to explore various attacks and devise multiple defenses. Although it is very difficult to keep up with the pace of results in this area, there are many excellent surveys on the topic. For instance, the surveys [1, 78] cover many interesting instances for which adversarial examples exist. In this chapter, we overview as well some of the most important findings regarding adversarial examples for DNNs. However, we adopt a different approach. Instead of creating a catalog of existing attacks and defenses, we present an adequately general framework which can recover many existing attacks. Theoretical findings regarding the nature of adversarial examples are additionally addressed. In this light, we address three problems in this chapter, namely, adversarial attacks, their theoretical explanation and adversarial defenses.

The first question is about generating adversarial examples and designing attacks. This is discussed in the first part of this chapter. Historically these examples were first found for classification tasks and were based on first order approximations of DNNs. These methods require knowledge of model parameters and are therefore sometimes called white-box attacks. We overview some of the most important attacks including



**Fig. 1** A demonstration from [24] of adversarial examples generated using the FGSM. By adding an imperceptibly small vector, we can change GoogLeNet’s classification of the image

iterative and non-iterative methods, as well as single and multiple pixel attacks. Instead of listing different attacks, our goal is to present a unifying framework for generating adversarial examples. The framework, which goes beyond classification problems, is based on a convex optimization formulation of adversarial input generation. We overview, furthermore, black-box attacks where only partial knowledge of model parameters is available for generating adversarial examples. Universal adversarial perturbations and the transferability of adversarial examples are other topics discussed in this part.

The second question is about the nature of adversarial examples. Why are DNNs and other machine learning models vulnerable to adversarial examples? In the second part, we overview some of the attempts to investigate theoretically this question. In many works, the adversarial vulnerability is attributed to some properties of machine learning models. Some examples are linearity of models, curvature of decision boundaries of classifiers and low  $\ell_1$ -norm of weight matrices. After reviewing some of these theories, we discuss statistical learning theoretic approaches that explore the relation between adversarial robustness and generalization capabilities of machine learning models. Out of this study come new guidelines for designing adversarially robust algorithms, which brings us to the third question of this chapter. How can we design effective defenses against adversarial examples?

The defenses take up different approaches from modifying the training process by changing the training set to adding new regularizations or considering new DNN architectures or a combination of preceding approaches. Some of the most recent contributions in this direction are discussed in the last part.

## 1.1 Notation and Preliminaries

We introduce first the notation used in this chapter and some of the basic definitions needed throughout this chapter. The letters  $\mathbf{x}, \mathbf{y}, \dots$  are used for vectors,  $\mathbf{A}, \mathbf{B}, \dots$  for matrices and  $\mathcal{X}, \mathcal{Y}, \dots$  for sets. We denote the set  $\{1, \dots, n\}$  by  $[n]$  for  $n \in \mathbb{N}$ . For any vector  $\mathbf{x} = (x_1, \dots, x_n)^\top \in \mathbb{R}^n$  and  $p \in \mathbb{N}$ , the  $\ell_p$ -norm of  $\mathbf{x}$  is defined by

$$\|\mathbf{x}\|_p := \left( \sum_{i=1}^n x_i^p \right)^{1/p}.$$

When  $p$  tends to zero, the above definition converges to the number of non-zero entries of the vector. This is called, with an abuse of terminology, the  $\ell_0$ -norm. The explicit definition is given as

$$\|\mathbf{x}\|_0 := \sum_{i=1}^n \mathbf{1}(x_i \neq 0).$$

The  $\ell_0$ -norm gives the sparsity order of the vector  $\mathbf{x}$ . The  $\ell_\infty$ -norm of a vector  $\mathbf{x}$  is obtained when  $p \rightarrow \infty$ . It is defined as

$$\|\mathbf{x}\|_\infty := \max_{i \in [n]} |x_i|.$$

The norm of a matrix  $\mathbf{X} \in \mathbb{R}^{m \times n}$  is similarly defined. The Frobenius norm of  $\mathbf{X}$  is denoted by  $\|\mathbf{X}\|_F$  and defined as:

$$\|\mathbf{X}\|_F := \left( \sum_{i=1}^m \sum_{j=1}^n X_{ij}^2 \right)^{1/2}.$$

The Shatten  $p$ -norm of the matrix  $\mathbf{X}$  is equal to the  $\ell_p$ -norm of the singular value vector  $(\sigma_1, \dots, \sigma_{\min(m,n)})$  of  $\mathbf{X}$ , namely:

$$\|\mathbf{X}\|_p := \left( \sum_{i=1}^{\min\{m,n\}} \sigma_i^p \right)^{1/p}.$$

The Frobenius norm of  $\mathbf{X}$  is equivalent to the  $\ell_2$ -norm of the singular value vector. The  $\ell_1$ -norm of the singular value vector is called the nuclear norm. The  $\ell_0$ -norm is similarly defined and gives the rank of  $\mathbf{X}$ .

Consider a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  given by  $f(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$  for  $m$  function  $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ . The Jacobian of  $f$  at  $\mathbf{x}$  is denoted by  $J_f(\mathbf{x})$  and defined as

$$J_f(\mathbf{x}) := \left( \frac{\partial f}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial f}{\partial x_m}(\mathbf{x}) \right) = \left[ \frac{\partial f_i}{\partial x_j}(\mathbf{x}) \right]_{i \in [n], j \in [m]}.$$

## 2 Adversarial Perturbation Design

Adversarial attacks follow ubiquitously the same pattern. An adversarial attacker is assumed to have access to the system input. This can be the input of DNNs. It applies perturbations to the system inputs under an additional and important constraint. The perturbations should be restricted in some sense. For image-based tasks, this means that an ordinary observer should not be capable of spotting, at least immediately, a significant change in the image and its label. More generally, this constraint makes it hard for the administrator to detect the perturbations. Finally, and most importantly, the system performance, for example its classification accuracy, should be severely degraded. The attacks in [24, 47, 57] follow similar guidelines. Two categories of adversarial attacks can be envisaged, white-box and black-box attacks. In white-box attacks, the architecture of the target algorithms are known to the attacker, although there are attacks with only partial knowledge of the architecture. In contrast stand

black-box attacks, which require no information about the target neural network, see for instance [62].

In the pioneering work of [69], the attack is based on finding adversarial perturbations that maximize the prediction error at the output. The perturbations are approximated by minimizing the  $\ell_2$ -norm of the perturbation. If the multi-class classifier mapping is defined by  $f : \mathbb{R}^n \rightarrow [K]$ , Szegedy et al. [69] minimize the  $\ell_2$ -norm of the perturbation  $\eta$  such that the classifier output is changed to the target label  $l \in [K]$ , i.e.,  $f(\mathbf{x} + \eta) = l$ . The perturbation  $\eta$  is constrained to be inside the box  $[0, 1]^n$ . The adversarial example is obtained by adding the perturbation  $\eta$  to the input vector  $\mathbf{x}$ . In the next attack, the FGSM in [24], the sign of the gradient of the cost function is used for designing perturbations which were scaled to have bounded  $\ell_\infty$ -norm, and therefore to be almost undetectable. If the cost function used for training is given by  $c(\mathbf{x})$ , the perturbation is given by  $\eta = \epsilon \text{sign}(\nabla c(\mathbf{x}))$ . The  $\ell_\infty$ -norm of the perturbation is  $\epsilon$ . An example of the FGSM is shown in Fig. 1. Iterative procedures or randomizations can significantly strengthen adversarial attacks. An iterative linearization of the DNN is proposed in the algorithm DeepFool [47] to generate minimal  $\ell_p$ -norm perturbations for  $p > 1$ . The iterative approach continues to add perturbations with bounded  $\ell_p$ -norm until the classifier's output is altered. An iterative version of FGSM, called Basic Iterative Method (BIM) is proposed in [35]. The Projected Gradient Descent (PGD) attack is an extension of previous techniques, proposed in [43], where randomness is additionally introduced in the computation of adversarial perturbations. The PGD attack can bypass many defenses and is employed in [43] to devise a defense against adversarial examples. An iterative algorithm based on PGD combined with randomization is introduced in [5] and has been used to dismantle many defenses so far [4]. Another popular way of generating adversarial examples is by constraining the  $\ell_0$ -norm of the perturbation. Manipulating only few entries, these types of attacks are known as single pixel attacks [66] and multiple pixel attacks [52].

In what follows, to generate adversarial examples, we provide a unifying framework that incorporates the above techniques. The main ingredient of this framework is perturbation analysis. Given a classifier function, the perturbation analysis of this function quantifies how much its output is perturbed when a known perturbation is applied to its input. An approximation of this output error is usually obtained using a first-order Taylor approximation of the function, under the assumption that the input perturbations are of small norms. Adversarial examples suitably fall into this framework, as they are perturbed versions of original inputs, the perturbations are small and the function at hand comes naturally from the model. Consider, for example, the FGSM given in [24]. The proposed attack aims at maximizing the training loss function that is approximated by its first-order Taylor approximation. Similarly, the authors of [24, 47] constructed adversarial examples by maximizing the error, on a relevant function, that occurs as a consequence of input perturbations. Iterative methods like the DeepFool method [47], the BIM [35], the PGD method [43], and the gradient-based norm-constrained method (GNM) [8], maximize the output perturbation using successive first order approximations. A summary about the connections

and differences between these methods is provided in [8]. It is based on this framework that we formulate the problem of generating adversarial examples in this section.

Let us first fix the terminology used in this section. The input of classifiers is denoted by  $\mathbf{x}$ . Then, adversarial examples are constructed by adding an adversarial perturbation  $\boldsymbol{\eta}$ , of the same dimension as  $\mathbf{x}$ , to that input. For a multi-class classification with  $K$  classes, a classifier maps inputs to the discrete set of labels  $[K]$ . Classifiers modeled by DNNs based their decision usually on a set of functions, often differentiable, known as *score functions*. These functions can replace the non-differentiable classification function for which a first-order Taylor approximation is not possible because the gradients are not properly defined. The score functions and classification functions are defined below.

**Definition 1** (*Score functions and classifier functions*) A classifier is defined by the mapping  $k : \mathbb{R}^M \rightarrow [K]$  that maps an input  $\mathbf{x} \in \mathbb{R}^M$  to its estimated class  $k(\mathbf{x}) \in [K]$ . The mapping  $k(\cdot)$  is itself defined by

$$k(\mathbf{x}) = \operatorname{argmax}_{l \in [K]} \{f_l(\mathbf{x})\}, \quad (1)$$

where  $f_l(\mathbf{x}) : \mathbb{R}^M \rightarrow \mathbb{R}$ 's represent the probability of class belonging. The function  $f(\mathbf{x})$  given by the vector  $(f_1(\mathbf{x}), \dots, f_K(\mathbf{x}))^\top$  is known as score function and can be assumed to be differentiable almost everywhere for many classifiers.

Finding adversarial examples amounts to finding a perturbation that changes the classifier's output. However, since they are imperceptible, such adversarial perturbations should not modify the inputs significantly. The undetectability of adversarial examples can be better understood using image classification tasks as an example. For instance, in Fig. 1 we observe that the human eye can not distinguish between the original and adversarial image. A common way to impose this restriction is by constraining adversarial perturbation to belong to a certain set of unnoticeable perturbations. For example, the authors of the FGSM bounded the  $\ell_\infty$ -norm of their perturbation, or in the DeepFool method, the norm is incrementally increased until the output classifier changes. Note that DeepFool may produce perceptible perturbations, while the FGSM may not fool the classifier.

Another way of imposing undetectability of adversarial examples is to impose on the input perturbation to preserve the outcome of the ground truth classifier [74], also known as oracle classifier. In many applications, the oracle classifier refers to the human brain. Similar to Definition 1, denote the score function of the oracle classifier as  $g : \mathbb{R}^M \rightarrow \mathbb{R}^K$ , which outputs a vector with entries  $g_l : \mathbb{R}^M \rightarrow \mathbb{R}$  for  $l = 1, \dots, K$ . The adversarial perturbation  $\boldsymbol{\eta}$  is said to be undetectable if

$$L_g(\mathbf{x}, \boldsymbol{\eta}) = g_{k(\mathbf{x})}(\mathbf{x} + \boldsymbol{\eta}) - \max_{l \neq k(\mathbf{x})} g_l(\mathbf{x} + \boldsymbol{\eta}) > 0. \quad (2)$$

Using this notion, the problem of finding adversarial examples amounts to the following.

**Definition 2** (*Adversarial Generation Problem*) For a given  $\mathbf{x} \in \mathbb{R}^M$ , the adversarial generation problem consists of finding a perturbation  $\boldsymbol{\eta} \in \mathbb{R}^M$  to fool the classifier  $k(\cdot)$  by the adversarial sample  $\hat{\mathbf{x}} = \mathbf{x} + \boldsymbol{\eta}$  such that  $k(\mathbf{x}) \neq k(\hat{\mathbf{x}})$  and the oracle classifier is not changed, i.e.,

$$\begin{aligned} \text{Find : } & \boldsymbol{\eta} \\ \text{s.t. } & L_f(\mathbf{x}, \boldsymbol{\eta}) = f_{k(\mathbf{x})}(\mathbf{x} + \boldsymbol{\eta}) - \max_{l \neq k(\mathbf{x})} f_l(\mathbf{x} + \boldsymbol{\eta}) < 0 \\ & L_g(\mathbf{x}, \boldsymbol{\eta}) = g_{k(\mathbf{x})}(\mathbf{x} + \boldsymbol{\eta}) - \max_{l \neq k(\mathbf{x})} g_l(\mathbf{x} + \boldsymbol{\eta}) > 0 \end{aligned} \quad (3)$$

However, since the oracle classifier is usually unknown, this problem is not interesting for practical purposes. To overcome this issue, it is shown in forthcoming sections how the solution of this problem can be approximated by tractable relaxations.

## 2.1 White-Box Attacks

The white-box setting corresponds to the scenario when the classification function  $f(\cdot)$  and input  $\mathbf{x}$  are both known to the attacker. Thus, adversarial perturbations are designed with full knowledge of the target system.

**Non-iterative Methods** As discussed above, the constraint on the oracle function of (3) cannot be computed in practice, since the oracle classifier is not available. To address this problem, such constraints are approximated by restricting the set of possible adversarial perturbations to a known subset. The most common choice is to restrict  $\boldsymbol{\eta}$  to belong to the set of vectors with bounded  $\ell_p$ -norm for  $p \geq 1$ . The values of  $p$  are restricted to be  $p \geq 1$  so that the set  $\|\boldsymbol{\eta}\|_p \leq \epsilon$  is convex for any  $\epsilon > 0$ . Note that the choice of  $p$  will determine the structure of the obtained adversarial examples. The case of  $p = \infty$  has been the focus of research in recent years. Even after replacing the oracle constraint on (3), with a convex one, the problem remains non-convex. For the case of white-box attacks, a similar relaxation can be carried out by approximating  $L_f(\mathbf{x}, \cdot)$  with its first-order Taylor expansion. This is possible since we assume to have full knowledge about  $\mathbf{x}$  and the function  $f(\cdot)$ .

To that end, the first-order Taylor expansion of  $L_f(\mathbf{x}, \cdot)$  around  $\mathbf{0}$  leads to

$$L_f(\mathbf{x}, \boldsymbol{\eta}) = L_f(\mathbf{x}, \mathbf{0}) + \boldsymbol{\eta}^\top \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \mathbf{0}) + \mathcal{O}(\|\boldsymbol{\eta}\|_2^2),$$

where  $\mathcal{O}(\|\boldsymbol{\eta}\|_2^2)$  contains higher order terms. Therefore, by replacing the oracle function constraint in (3) with  $\|\boldsymbol{\eta}\|_p \leq \epsilon$ , for sufficiently small  $\epsilon \in \mathbb{R}^+$ , we get

$$\begin{aligned} \text{Find: } & \boldsymbol{\eta} \\ \text{s.t. } & L_f(\mathbf{x}, \mathbf{0}) + \boldsymbol{\eta}^\top \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \mathbf{0}) < 0, \quad \|\boldsymbol{\eta}\|_p \leq \epsilon, \end{aligned} \quad (4)$$

which is a relaxed version of the problem exposed in (3). This formulation of the problem can be used to construct well known existing adversarial attacks from the literature. This will be discussed in detail in Sect. 2.1. Nevertheless, the following theorem shows that this problem is not always feasible.

**Theorem 1** *The optimization problem (4) is not feasible iff for  $q = \frac{p}{p-1}$*

$$\epsilon \|\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})\|_q < L_f(\mathbf{x}, \mathbf{0}). \quad (5)$$

The proof can be obtained from the results in [27], as well as in [7].

The theorem points to the insight that there might be no perturbation that is small enough and yet changes the output label. This implication befits the intuition that it should not be expected to fool a classifier for an arbitrarily small  $\epsilon$  with the perturbation's norm constraint. This result suggests that a feasible problem can be obtained if we only impose one of the constraints while trying to preserve the other one as much as possible. To that end, a proper objective function that penalizes the deviation from the original constraint is minimized. This gives rise to the following two problems, as feasible counterparts of (4).

First, the norm-constraint in (4) is imposed resulting in the following optimization problem, called GNM in [7]. It minimizes  $L_f(\mathbf{x}, \mathbf{0}) + \eta^T \nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})$  as

$$\min_{\eta} L_f(\mathbf{x}, \mathbf{0}) + \eta^T \nabla_{\eta} L_f(\mathbf{x}, \mathbf{0}) \quad \text{s.t.} \quad \|\eta\|_p \leq \epsilon. \quad (6)$$

Using this approach we can find the best possible perturbation under the norm-constraint. However, a proper value for  $\epsilon$  must be chosen beforehand to guarantee that the perturbations remain unnoticed. Moreover, this problem has a closed form solution which can be computed efficiently, as stated in the following theorem.

**Theorem 2** *If  $\nabla_{\eta} L_f(\mathbf{x}, \eta) = (\frac{\partial L_f(\mathbf{x}, \eta)}{\partial \eta_1}, \dots, \frac{\partial L_f(\mathbf{x}, \eta)}{\partial \eta_M})$ , the closed form solution to the minimizer of the problem (6) is given by*

$$\eta = -\epsilon \frac{1}{\|\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})\|_q^{q-1}} \text{sign}(\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})) \odot |\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})|^{q-1} \quad (7)$$

for  $q = \frac{p}{p-1}$ , where  $\text{sign}(\cdot)$  and  $|\cdot|^{q-1}$  are applied element-wise, and  $\odot$  denotes the element-wise (Hadamard) product. Particularly for  $p = \infty$ , we have  $q = 1$  and the solution is given by the following

$$\eta = -\epsilon \text{sign}(\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})). \quad (8)$$

The proof can be found in [7]. One advantage of using (6), besides having a closed-form solution, is that additional constraints on the perturbation can be added to the problem. In addition, the solution shown in (7) can be reused for other choices of

$L_f(\mathbf{x}, \cdot)$ , which can be more suitable depending on the scenario. For instance, the FGSM chooses  $L_f(\mathbf{x}, \cdot)$  to be the negative of the loss function used for training, which is often the cross-entropy loss in classification problems. Then, minimizing  $L_f(\mathbf{x}, \cdot)$  corresponds to maximizing the loss. A caveat is that using problem (6) ensures perturbations with bounded norms, but such perturbations may not be able to fool the classifier.

A second approach for relaxing (4) into a feasible problem is to keep the constraint regarding  $L_f(\mathbf{x}, \cdot)$  and minimize over the norm of  $\boldsymbol{\eta}$ . Therefore, the problem of (4) is replaced by

$$\min_{\boldsymbol{\eta}} \|\boldsymbol{\eta}\|_p \quad \text{s.t.} \quad L_f(\mathbf{x}, \mathbf{0}) + \boldsymbol{\eta}^\top \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \mathbf{0}) \leq 0. \quad (9)$$

This approach is used by [47] on every iteration of the DeepFool algorithm (more details in Sect. 2.1). Similarly to (6), this problem has a closed form solution as well, which is given in the following theorem.

**Theorem 3** *If  $\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \boldsymbol{\eta}) = (\frac{\partial L_f(\mathbf{x}, \boldsymbol{\eta})}{\partial \eta_1}, \dots, \frac{\partial L_f(\mathbf{x}, \boldsymbol{\eta})}{\partial \eta_M})$ , the closed form solution to the problem (9) is given by*

$$\boldsymbol{\eta} = -\frac{L_f(\mathbf{x}, \mathbf{0})}{\|\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \mathbf{0})\|_q^{q-1}} \operatorname{sign}(\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \mathbf{0})) \odot |\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \mathbf{0})|^{q-1} \quad (10)$$

for  $q = \frac{p}{p-1}$ .

Observe that the perturbation from Theorem 3, similar to the solution in Theorem 2, is nothing but an adjusted version of the gradient of the classifier with a different norm. The perturbation in (10) might grow unbounded to ensure that the classifier is misled, which makes it perceptible by the oracle. There are other similar methods for computing adversarial examples that depend on a first-order approximation of other performance-related functions. These algorithms are later shown to be slight variations of the methods presented in this section. Furthermore, using the present formulation, we can build iterative procedures by repeating the optimization problem until the classifier output changes. In Sect. 2.1, we compare different methods, which are formulated as iterative versions of (6) and (9).

There are some methods that rely on adding randomness in the generation process. The PGD attack, from [43], is one well known example. For the PGD attack, the first-order approximation is taken not around  $\boldsymbol{\eta} = \mathbf{0}$ , but instead, around a random point  $\tilde{\boldsymbol{\eta}}$  with its norm bounded by some  $\tilde{\epsilon}$ , that is  $\tilde{\epsilon} \triangleq \|\tilde{\boldsymbol{\eta}}\|_p \leq \epsilon$ . In short, the objective function  $L_f(\mathbf{x}, \cdot)$  is approximated by its linear counterpart around the point  $\tilde{\boldsymbol{\eta}}$ , which lies within an  $\tilde{\epsilon}$ -radius from  $\boldsymbol{\eta} = \mathbf{0}$ . The distribution  $\tilde{\boldsymbol{\eta}}$  can be arbitrarily chosen as long as the norm constraint is not violated. A common choice is to use the uniform distribution over the set of vectors with bounded  $\ell_p$  norm. We denote this technique as *dithering*. Moreover, displacing the center of the first order approximation from  $\mathbf{0}$  to  $\tilde{\boldsymbol{\eta}}$  does not lead to solutions which differ from the ones given so far. This is true since  $L_f(\mathbf{x}, \boldsymbol{\eta}) \approx L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}) + (\boldsymbol{\eta} - \tilde{\boldsymbol{\eta}})^\top \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}})$  leads to the following problem

$$\min_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}) + (\boldsymbol{\eta} - \tilde{\boldsymbol{\eta}})^T \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}) \quad \text{s.t.} \quad \|\boldsymbol{\eta}\|_p \leq \epsilon,$$

which corresponds to solving

$$\min_{\boldsymbol{\eta}} \boldsymbol{\eta}^T \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}) \quad \text{s.t.} \quad \|\boldsymbol{\eta}\|_p \leq \epsilon. \quad (11)$$

When training models with adversarial examples, it is advantageous to add randomness to their computation in order to increase the diversity of the adversarial perturbations during the training [72], as done with dithering technique in quantization literature. Further details about training with adversarial examples are discussed in Sect. 4.1.

**Single Subset Attacks.** In the field of image recognition, it is also common to model undetectability by restricting the number of pixels that can be altered by an attacker. Single and multiple pixel attacks are introduced to this end. For the case of gray-scale images, altering only one value of the input vector is equivalent to a single pixel attack. This is, however, not a general rule. If inputs are RGB images, each pixel will be defined as a subset of three values.

Since adversarial attacks go beyond image based systems, we allude as single subset attacks to those whose target is only a subset of entries. Given that perturbations belong to  $\mathbb{R}^M$ , let us partition  $[M] = \{1, \dots, M\}$  into  $S$  possible subsets  $\mathcal{S}_1, \dots, \mathcal{S}_S$ .

These sets may be of different size, but for the sake of clarity let us assume that they have the same cardinality  $Z = M/S$ , where  $\mathcal{S}_s = \{i_s^1, \dots, i_s^Z\} \subseteq [M]$ . Define the mixed zero- $\mathcal{S}$  norm  $\|\cdot\|_{0,\mathcal{S}}$  of a vector, for the partition  $\mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_S\}$ , as the number of subsets including at least one index related to a non-zero entry of  $\mathbf{x}$ ,<sup>1</sup> that is

$$\|\mathbf{x}\|_{0,\mathcal{S}} = \sum_{i=1}^S \mathbf{1}(\|\mathbf{x}_{\mathcal{S}_i}\| \neq 0),$$

where  $\mathbf{1}(\cdot)$  denotes the indicator function. Hence, the norm  $\|\boldsymbol{\eta}\|_{0,\mathcal{S}}$  counts the number of subsets altered by an attacker. Moreover, we can guarantee that only one subset stays active by including this as an additional constraint in (3), yielding

$$\min_{\boldsymbol{\eta}} L_f(\mathbf{x}, \boldsymbol{\eta}) \quad \text{s.t.} \quad \|\boldsymbol{\eta}\|_{\infty} \leq \epsilon, \quad \|\boldsymbol{\eta}\|_{0,\mathcal{S}} = 1. \quad (12)$$

As a remark, the mixed norm  $\|\cdot\|_{0,\mathcal{S}}$  is extensively used in signal processing and compressed sensing to promote group sparsity [57]. In a similar manner as in Sect. 2.1, we employ the approximation  $L_f(\mathbf{x}, \boldsymbol{\eta}) \approx L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}) + (\boldsymbol{\eta} - \tilde{\boldsymbol{\eta}})^T \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}})$  which yields the following linear programming formulation of (12) as

$$\min_{\boldsymbol{\eta}} \boldsymbol{\eta}^T \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}) \quad \text{s.t.} \quad \|\boldsymbol{\eta}\|_{\infty} \leq \epsilon, \quad \|\boldsymbol{\eta}\|_{0,\mathcal{S}} = 1. \quad (13)$$

---

<sup>1</sup>Similar to the so-called  $\ell_0$ -norm, this is not a proper norm.

Given a subset  $\mathcal{S}_s$  we define  $\boldsymbol{\eta}_s$  as

$$\boldsymbol{\eta}_s = \operatorname{argmin}_{\boldsymbol{\eta}} \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}})^T \boldsymbol{\eta} \text{ s.t. } \|\boldsymbol{\eta}\|_\infty \leq \epsilon, (\boldsymbol{\eta})_{i_s^z} = 0 \quad \forall i_s^z \notin \mathcal{S}_s.$$

Note that we have a closed form solution for  $\boldsymbol{\eta}_s$ , that is

$$\boldsymbol{\eta}_s = -\epsilon \sum_{z=1}^Z \operatorname{sign}((\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}))_{i_s^z}) \mathbf{e}_{i_s^z},$$

which implies that  $\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}})^T \boldsymbol{\eta}_s = - \sum_{z=1}^Z |(\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}))_{i_s^z}|$ . Then, this problem has the closed form solution given by

$$\boldsymbol{\eta}^* = \boldsymbol{\eta}_{s*}, \text{ with } s^* = \operatorname{argmax}_s \sum_{z=1}^Z |(\nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}))_{i_s^z}|. \quad (14)$$

**Iterative Methods and Randomization** In the previous section, we summarized different versions of the problem of generating adversarial perturbations. An overview of these methods is shown in Table 1. For this section, we will work with these solutions to design adversarial perturbations using iterative approximations. Since the principle behind the approaches in (6) and (9) is the same, we will only focus on (6). Nevertheless, it is trivial to extend the algorithms presented in this section to use (9) instead.

In Algorithm 1, an iterative method based on (6) is introduced. This iterative version of (6) resembles a gradient descent method for minimizing  $L_f(\mathbf{x}, \boldsymbol{\eta})$  over  $\boldsymbol{\eta}$  with a fixed number of iterations and steps of equal  $\ell_p$ -norm. For that purpose, a set of parameters  $\tilde{\epsilon}_1, \dots, \tilde{\epsilon}_T$  is required to control the norm of random noise used for dithering. There is no dithering if  $\tilde{\epsilon}_i$  is set to zero for all  $i = 1, \dots, T$ . The well known PGD attack uses dithering by applying it at the initial iteration. In this attack  $\tilde{\epsilon}_2 = \dots = \tilde{\epsilon}_T = 0$ , and  $\operatorname{random}(\tilde{\epsilon}_1)$  generates a random vector with a uniform distribution over the  $\ell_p$ -ball of radius  $\tilde{\epsilon}_1$  (centered at  $\mathbf{0}$ ).

---

**Algorithm 1** Iterative extension for  $\ell_p$  constrained methods.

---

```

input:  $\mathbf{x}, f(\cdot), T, \epsilon, \tilde{\epsilon}_1, \dots, \tilde{\epsilon}_T$ .
output:  $\boldsymbol{\eta}^*$ .
Initialize  $\boldsymbol{\eta}_1 \leftarrow \mathbf{0}$ .
for  $t = 1, \dots, T$  do
     $\tilde{\boldsymbol{\eta}}_t \leftarrow \boldsymbol{\eta}_t + \operatorname{random}(\tilde{\epsilon}_t)$ 
     $\boldsymbol{\eta}_t^* \leftarrow \operatorname{argmin}_{\boldsymbol{\eta}} \boldsymbol{\eta}^T \nabla_{\boldsymbol{\eta}} L_f(\mathbf{x}, \tilde{\boldsymbol{\eta}}_t) \text{ s.t. } \|\boldsymbol{\eta}\|_p \leq \epsilon/T$  (Table 1)
     $\boldsymbol{\eta}_{t+1} \leftarrow \boldsymbol{\eta}_t + \boldsymbol{\eta}_t^*$ 
end for
return:  $\boldsymbol{\eta}^* \leftarrow \boldsymbol{\eta}_T$ 

```

---

**Table 1** Summary of the obtained closed-form solutions

Type of attack	Relaxed problem	Closed-form solution
$\ell_2 / \ell_\infty$ constrained	(6)	(7)
Single-subset attack	(13)	(14)

**Table 2** Recovering existing attacks in classification using this framework [8]

Algorithm	Objective function $L$	Iterative	Dithering
FGSM [24]	Cross-entropy	✗	✗
DeepFool [47]	(2) with $l$ chosen using $\hat{\rho}_1(f)$	✓	✗
BIM [35]	Cross-entropy	✓	✗
PGD [43]	Cross-entropy	✓	✓
Targeted	(2) with $l$ fixed to the target	✓	✓
GNM [7]	(2)	✓	✓
Regression [8]	$\ell_2$ -norm of output perturbation	✓	✓

Finally, computing  $\boldsymbol{\eta}_t^*$  with the additional constraint  $\|\boldsymbol{\eta}\|_{0,\mathcal{S}} = 1$  in Algorithm 1 leads to a multiple subset attack. For such an attack one must additionally subtract previously modified subsets from  $\mathcal{S}$ . This results in a new subset being altered at every iteration. Similarly, given a class label  $\bar{l} \in [K]$ , changing the objective function to

$$L_f(\mathbf{x}, \boldsymbol{\eta}) = f_{k(\mathbf{x})}(\mathbf{x} + \boldsymbol{\eta}) - f_{\bar{l}}(\mathbf{x} + \boldsymbol{\eta})$$

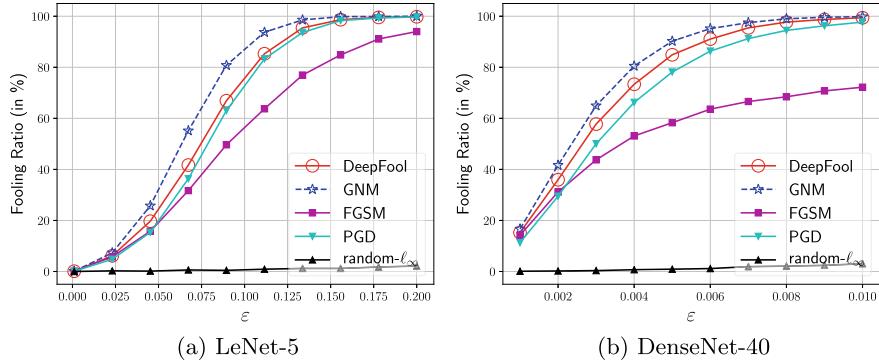
leads to a *targeted* attack, that is when the objective is to apply perturbations such that the outcome of classification is always some “target” class  $\bar{l}$ .

As we can see, different configurations for Algorithm 1 lead to known adversarial attacks from the literature. A summary of Algorithm 1 configurations with their corresponding attack from the literature is presented in Table 2. In classification, these methods are usually compared using the *fooling ratio*, that is the percentage of correctly classified inputs that are misclassified when adversarial perturbations are added. Visualizing the fooling ratio for different values of  $\epsilon$  is often used to empirically asses the performance of an adversarial attack. For example, in Fig. 2 we observe the fooling ratio of different attacks on standard DNNs (not trained to resist adversarial attacks).

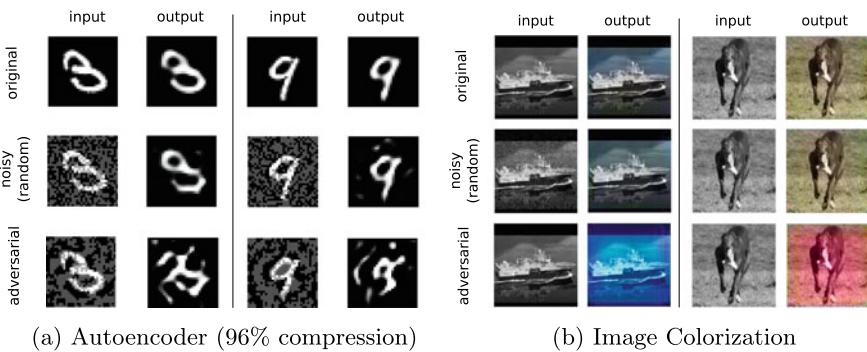
**Regression Problems and Other Learning Tasks.** The objective

$$L_f(\mathbf{x}, \boldsymbol{\eta}) = -\|f(\mathbf{x} + \boldsymbol{\eta}) - f(\mathbf{x} + \boldsymbol{\eta})\|_2$$

can be used to attack regression problems as well. This possibility is investigated in [8], where the objective is to perturb the output of a regression model as much as



**Fig. 2** Fooling ratio, from [7], of different adversarial attacks on vanilla DNNs on the MNIST dataset. **a** 5-layered LeNet architecture from [37], **b** DenseNet architecture from [30] with 40 layers



**Fig. 3** Adversarial examples for regression [8]. **a** MNIST autoencoder, **b** STL-10 colorization network

possible. Two examples are provided using autoencoders and colorization<sup>2</sup> DNNs in Fig. 3. In that figure, we observe how adversarial perturbations heavily distort the outcome of regression. Using the principles explicated in this section, other algorithms have been developed for attacking other types of learning systems. In the field of computer vision, [28] constructed an attack on image segmentation, while [76] designed attacks for object detection. The Houdini attack [12] aims at distorting speech recognition systems. In addition, [53] tailored an attack for recurrent neural networks, and [40] for reinforcement learning. Adversarial examples exist for probabilistic methods as well. For instance, [33] showed the existence of adversarial examples for generative models. For regression problems, [70] designed an attack that specifically targets variational autoencoders.

**Robustness metrics.** Going back to the definitions of Sect. 2.1, Theorem 1 shows that given a vector  $\mathbf{x}$  and a score function  $f(\cdot)$ , the adversarial perturbation should have at least  $\ell_p$ -norm equal to  $\frac{L_f(\mathbf{x}, \mathbf{0})}{\|\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})\|_q}$  to fool the linearized version of  $f(\cdot)$ . In

<sup>2</sup>A colorization model predicts the color values for every pixel in a given gray-scale image.

**Table 3** Experiment from [7] showing the robustness measures for different DNNs on the MNIST and CIFAR-10 datasets. The acronyms FCNN denotes a standard fully connected neural network, while NIN refers to the network-in-network architecture from [39]. LeNet-5 and DenseNet are the same architectures used in Fig. 2

	Test error (%)	$\hat{\rho}_1(f)$ [47]	$\hat{\rho}_2(f)$ [7]	Fooled >99%
FCNN (MNIST)	1.7	0.036	0.034	$\epsilon = 0.076$
LeNet-5 (MNIST)	0.9	<b>0.077</b>	<b>0.061</b>	$\epsilon = 0.164$
NIN (CIFAR-10)	13.8	<b>0.012</b>	<b>0.004</b>	$\epsilon = 0.018$
DenseNet (CIFAR-10)	5.2	0.006	0.002	$\epsilon = 0.010$

other words if the ratio  $\frac{L_f(\mathbf{x}, \mathbf{0})}{\|\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})\|_q}$  is small, then it is easier to fool the network with  $\ell_p$ -attacks. In that sense, Theorem 1 provides an insight into the stability of classifiers. Therefore, regularizing the loss function with  $\frac{L_f(\mathbf{x}, \mathbf{0})}{\|\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})\|_q}$  may lead to adversarial robustness. Moreover, one can also include dithering by regularizing with  $\frac{L_f(\mathbf{x}, \tilde{\eta})}{\|\nabla_{\eta} L_f(\mathbf{x}, \tilde{\eta})\|_q}$  with some randomly chosen  $\tilde{\eta}$ .

In [47], the authors suggest that the robustness of the classifiers can be measured as

$$\hat{\rho}_1(f) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \frac{\|\hat{\mathbf{r}}(\mathbf{x})\|_p}{\|\mathbf{x}\|_p},$$

where  $\mathcal{D}$  denotes the test set and  $\hat{\mathbf{r}}(\mathbf{x})$  is the minimum perturbation required to change the classifier's output. Proposition 1 suggests that one can also use the following as the measure of robustness

$$\hat{\rho}_2(f) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \frac{L_f(\mathbf{x}, \mathbf{0})}{\|\nabla_{\eta} L_f(\mathbf{x}, \mathbf{0})\|_q}.$$

The lower  $\hat{\rho}_2(f)$ , the easier it gets to fool the classifier and therefore it becomes less robust to adversarial examples. According to the experiments in [7], shown in Table 3, these two robustness metrics seem to be coherent when measuring the robustness of non-adversarially trained DNNs.

## 2.2 Black-Box Attacks and Universal Adversarial Perturbations

So far we have assumed that the adversarial attacker has perfect knowledge of the target classifier function  $f(\cdot)$  as well as the input  $\mathbf{x}$ . By loosening of these requirements, into more realistic assumptions, new types of algorithms arise, namely

- **black-box attacks:** these methods correspond to the settings where the target classifier  $f(\cdot)$  is unknown but the input  $\mathbf{x}$  may still be known to the attacker,
- **universal adversarial perturbations:** these perturbations are designed to work regardless of the input  $\mathbf{x}$ , which is assumed to be unknown. Nevertheless, the classifier  $f(\cdot)$  may be available to the attacker.

If both the target model  $f(\cdot)$  and input  $\mathbf{x}$  are unknown to the attacker, the adversarial attack would be a black-box as well as universal adversarial perturbation. These types of attacks are still possible by assuming partial or indirect knowledge about the input  $\mathbf{x}$  and the classifier  $f(\cdot)$ . For example, the attacker may have access to a set of independent realizations  $\{\mathbf{x}_1, \mathbf{x}_2, \dots\}$  of the input, which provides knowledge about the input distribution  $P_{\mathbf{x}}$ . Similarly, implicit information about the classifier can be inferred by observing the independent realizations of the pairs  $\{(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \dots\}$ . Finally, we may also have knowledge about the structure (number of layers, types of connections, activation functions, etc.) of the used classifier.

It is probably unexpected that some adversarial perturbations produce the same effect over different inputs and different DNNs architectures, although they are generated for a particular model. These universal adversarial perturbations are reported in [24, 57] where the authors show the existence of such perturbations for various datasets and DNNs. This phenomena suggests that there exist certain common properties shared by adversarial perturbations that account for most of the success when attacking a system. This can explain why certain perturbations are able simultaneously fool a target DNN on different inputs. Adversarial examples are indeed transferable. In [72] the authors construct an attack such that adversarial examples can transfer from one random instance of a neural network to another. Surprisingly, these methods were proved to be effective against well known DNNs. Since no explicit knowledge about the DNN weights is required to compute these perturbations, they can be thought of as black-box attacks. Moreover, the authors showed that including such black-box adversarial examples into the training set significantly enhances the robustness of neural networks. Finally, the authors in [45] showed that there exist adversarial examples that are both universal and black-box, that is perturbations that are independent from target DNN and input.

**Black-Box Attacks.** As discussed, in the black-box setting the classifier function  $f(\cdot)$  is unknown, thus we cannot compute the gradient necessary for Algorithm 1. A common approach to circumvent this issue is to estimate the gradient by choosing a substitute model  $\tilde{f}$  which is hoped to behave in a similar way as the unknown  $f(\cdot)$ . This concept is introduced in [51] under the assumption that the input  $\mathbf{x}$  is known, as well as  $n$  independent realizations of  $(\mathbf{x}, f(\mathbf{x}))$  denoted as  $(\mathbf{x}_1, f(\mathbf{x}_1)), \dots, (\mathbf{x}_n, f(\mathbf{x}_n))$ . This method consists on the following two steps:

1. Train a substitute model  $\tilde{f}$  that predicts  $f(\mathbf{x})$ , thus it resembles the target classifier.
2. Perform a white-box attack on the substitute model  $\tilde{f}$  and hope it transfers to the target model  $f(\cdot)$ .

This concept is later extended in [41], where the authors make use of several substitute models, that is  $\tilde{f}_1, \dots, \tilde{f}_r$  for  $r > 1$ . In that work, adversarial perturbations are computed by approximately solving the following optimization for an ensemble of loss functions:

$$\min_{\boldsymbol{\eta}} -\log \left( \sum_{i=1}^r \alpha_i \mathcal{L}_{\tilde{f}_i}(\mathbf{x}, \boldsymbol{\eta}) \right) + \lambda \|\boldsymbol{\eta}\|_p,$$

where  $\lambda > 0$ ,  $p > 1$ ,  $0 < \alpha_i < 1$ ,  $\sum_i \alpha_i = 1$  and  $\mathcal{L}_f(\mathbf{x}, \cdot)$  is some positive loss function like the cross-entropy loss of  $f(\cdot)$  at the point  $(\mathbf{x} + \boldsymbol{\eta})$ . The key idea of this method is that a perturbation  $\boldsymbol{\eta}$  that is able to fool the classifiers  $\tilde{f}_1, \dots, \tilde{f}_r$  will most likely fool the unknown classifier  $\tilde{f}_{r+1} \triangleq f$  as well. Note that it is also possible to generate norm-constrained versions of this method by approximately solving

$$\min_{\boldsymbol{\eta}} -\log \left( \sum_{i=1}^r \alpha_i \mathcal{L}_{\tilde{f}_i}(\mathbf{x}, \boldsymbol{\eta}) \right) \text{ s.t. } \|\boldsymbol{\eta}\|_p \leq \epsilon \quad (15)$$

using the same methods described in Table 1.

**Universal Adversarial Perturbations.** Given  $0 < \delta < 1$ , the paradigm of designing universal adversarial perturbations  $\mathbf{u}$  can be summarized as follows

$$\begin{aligned} \text{Find : } & \mathbf{u} \\ \text{s.t. } & \|\mathbf{u}\|_p \leq \epsilon \\ & P_{\mathbf{x}}(k(\mathbf{x} + \mathbf{u}) \neq k(\mathbf{x})) \geq 1 - \delta. \end{aligned}$$

Note that in order to approximately solve this problem one needs information about the distribution of the input. A common assumption when designing universal perturbations is that the attacker has perfect knowledge of the classifier  $k(\cdot)$ , but only partial knowledge about  $P_{\mathbf{x}}$  in the form of  $n$  independent realizations  $\mathcal{X}_n = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  of  $\mathbf{x} \sim P_{\mathbf{x}}$ .

This problem is first approached in [45] by iteratively aggregating the perturbations that move  $\mathbf{x}_1, \dots, \mathbf{x}_n$  to their corresponding decision boundaries. Given an input  $\mathbf{x}_i$ , such perturbations are computed by iteratively solving (9) in the same manner as in Algorithm 1. Then, in order to preserve the  $\ell_p$ -norm constraint, these perturbations are projected into the  $\ell_p$  ball of radius  $\epsilon$ . A summary of this method is shown in Algorithm 2. In addition, some example pictures showing the effectiveness of this algorithm are shown in Fig. 4. Note that this algorithm does not converge for an arbitrary choice of  $\delta$ , thus additional stopping criteria are needed.

**Algorithm 2** Universal adversarial perturbations with constrained  $\ell_p$ -norm.

---

```

input:  $\mathbf{x}_1, \dots, \mathbf{x}_n, k(\cdot), \epsilon, \delta$ .
output:  $\mathbf{u}^*$ .
Initialize  $\mathbf{u}^* \leftarrow \mathbf{0}$ .
while  $P_{\mathbf{x} \in \mathcal{X}_n}(k(\mathbf{x} + \mathbf{u}) \neq k(\mathbf{x})) \geq 1 - \delta$  do
    Shuffle  $\mathcal{X}_n$ 
    for  $i = 1, \dots, n$  do
        if  $k(\mathbf{x}_i + \mathbf{u}^*) = k(\mathbf{x}_i)$  then
            Compute the minimal perturbation that sends  $\mathbf{x}_i$  to the decision boundary:
             $\boldsymbol{\eta}^* \leftarrow \operatorname{argmin}_{\boldsymbol{\eta}} \|\boldsymbol{\eta}\|_2$  s.t.  $k(\mathbf{x}_i + \boldsymbol{\eta}^*) \neq k(\mathbf{x}_i)$ 
            Project the perturbation  $\boldsymbol{\eta}^*$  into the  $\ell_p$  ball of radius  $\epsilon$ :
             $\mathbf{u}^* \leftarrow \operatorname{argmin}_{\mathbf{u}} \|\mathbf{u}^* + \boldsymbol{\eta}^* - \mathbf{u}\|_2$  s.t.  $\|\mathbf{u}\|_p \leq \epsilon$ 
        end if
    end for
end while
return:  $\mathbf{u}^*$ 

```

---

In a similar fashion as in (15), a universal adversarial perturbation can be obtained by minimizing and ensemble of objective functions, that is

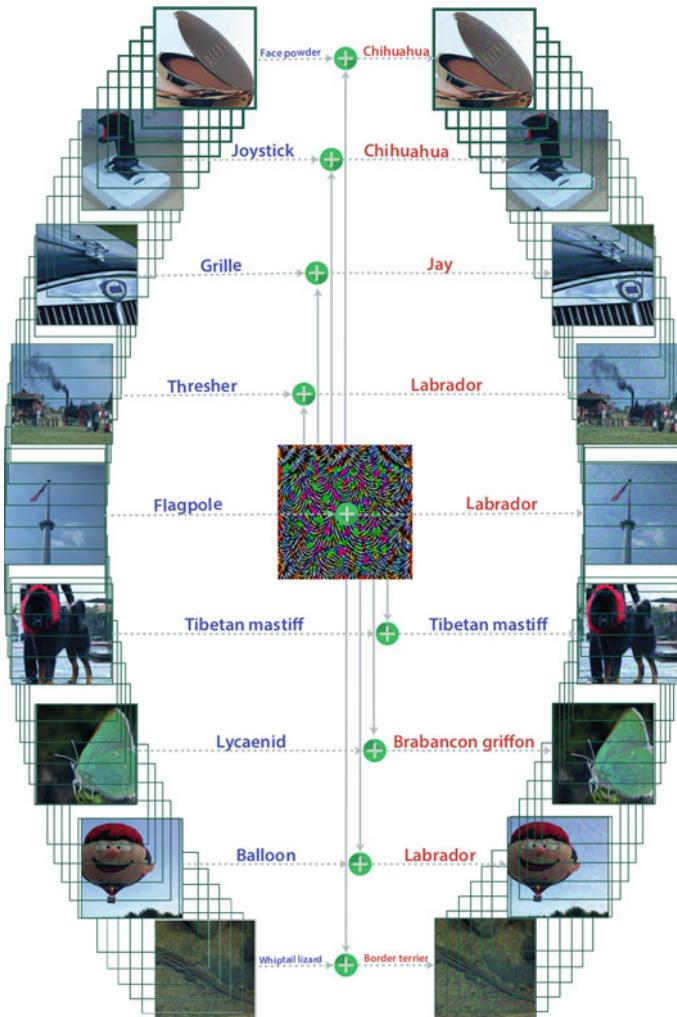
$$\min_{\boldsymbol{\eta}} \sum_{i=1}^n L_f(\mathbf{x}_i, \boldsymbol{\eta}) \text{ s.t. } \|\boldsymbol{\eta}\|_p \leq \epsilon,$$

where  $L_f(\mathbf{x}, \cdot)$  is some objective function as in (2). Choosing  $L_f$  to be  $L_f(\mathbf{x}, \boldsymbol{\eta}) = \|f(\mathbf{x}) - f(\mathbf{x} + \boldsymbol{\eta})\|_p^p$  and using the approximation  $\|f(\mathbf{x}) - f(\mathbf{x} + \boldsymbol{\eta})\|_p \approx \|J_f(\mathbf{x})\boldsymbol{\eta}\|_p$ , where  $J_f(\mathbf{x})$  denotes the Jacobian matrix of  $f(\cdot)$  at  $\mathbf{x}$ , the method proposed in [32] is obtained. This method is based on the insight that a perturbation that manages to fool several known inputs will most likely fool an unknown one as well.

### 3 Theoretical Explanations of the Nature of Adversarial Examples

Among various theories regarding the nature of adversarial examples, two directions can be singled out. One line of research focuses on local properties of classifiers, for example, decision boundaries of classifiers and their geometric properties. A notable example is the linearity hypothesis, proposed by the authors in [24], where the existence of adversarial images is attributed to the approximate linearity of classifiers. Another line of research tends to explain such phenomena by means of global properties of classifiers such as the topological dimension of their feature spaces or the sparsity of weight matrices in DNNs. We present some of the most important results along these two lines.

There is, however, another theoretical question raised in the literature. After some experimental results witnessed a seemingly opposing relation between adversarial



**Fig. 4** The authors in [45] add a universal perturbation (center image) that is able to mislead the classification of several images

robustness and generalization, researchers discussed formally the connection between generalization properties of DNNs and their adversarial robustness. The central question is whether adversarial robustness is realized only at the cost of worse generalization. This question is the subject of the last part of this section.

### 3.1 Linearity Hypothesis and Curvature of Decision Boundaries

How can the existence of adversarial examples be explained? The experimental results showed that adversarial examples are also misclassified by neural networks trained either on a different dataset or with different hyper-parameters. Therefore, this phenomenon cannot be attributed to overfitting to a particular model. It is first conjectured in [24] that, the adversarial examples exists because neural networks are well approximated, locally, by linear classifiers. This hypothesis, known as the linearity hypothesis, is supported by easy generations of adversarial examples using first order approximation of neural networks.

For deep neural networks, this claim is mainly experimentally substantiated. The attacks applied to the linear approximation of neural networks around an instance manage to effectively generate misclassified examples, and therefore, the effectiveness of first-order approximation attacks testifies, according to [24], to the linearity of these models.

To elucidate this claim, consider a linear classifier for binary classification tasks with the parameter  $\mathbf{w} \in \mathbb{R}^M$ . The classification rule is given simply by  $\text{sign}(\mathbf{w}^\top \mathbf{x})$ . In this linear setting, the FGSM provides the best  $\ell_\infty$ -bounded adversarial perturbation, which is given by  $\eta = -\epsilon \text{sign}(\mathbf{w})$ . The total perturbation caused at the output is  $\eta^\top \mathbf{w}$ , which is equal to  $-\epsilon \|\mathbf{w}\|_1$ . The value of  $\|\mathbf{w}\|_1$  can be as large as  $\sqrt{M}$  for  $\mathbf{w}$ 's with unit norm. Therefore, a small  $\ell_\infty$ -perturbation can be blown up by  $\sqrt{M}$ . Two conclusions can initially be drawn from this example. First, small input perturbations can incur large output perturbations for high dimensional linear classifiers. The authors in [24] argue accordingly for the existence of adversarial examples. High dimensional approximately linear classifiers can blow up small perturbations at their output. It is, according to [24], the approximate linearity of Convolutional Neural Networks (CNNs) that explains the existence of adversarial examples for the ImageNet classification problem. Although CNNs have many non-linearities, the parameters of CNNs after training are chosen so that the non-linearity of the architecture is diminished.

The second conclusion points to the  $\ell_1$ -norm of the parameter  $\mathbf{w}$  as the key to control the effectiveness of  $\ell_\infty$ -attacks, which can be used to design robust neural networks as we will see later. In general the  $\ell_q$ -norm of  $\mathbf{w}$  controls the output perturbation for  $\ell_p$ -attacks where  $(p, q)$  are dual to each other.<sup>3</sup> However, this example only shows that for some linear classifiers, small input perturbations changes significantly the output.

The linearity hypothesis did not remain unchallenged. After all, the non-linear machine learning algorithms were equally vulnerable to adversarial examples. In [59], adversarial examples were generated by imposing a similarity constraint between the hidden representation of the perturbed image in DNN and the hidden representation of an image from a different class. An optimization problem is

---

<sup>3</sup>We call the pair  $(p, q)$  dual if the corresponding norms are dual. In particular  $1/p + 1/q = 1$ .

used to minimize the difference between hidden representations with constraints on the perturbation. The method yielded adversarial images which differed from other adversarial images in that they did not rely, even implicitly, on linear approximations of the model. They could not be generated from linear approximations. The linearity hypothesis cannot explain the existence of these adversarial images.

For ImageNet classification problem with CNNs, the authors in [42] examined the linearity hypothesis by comparing  $f(\eta)$  and  $f(\mathbf{x} + \eta) - f(\mathbf{x})$ . These two values are equal for linear classifiers, thus it is used to measure the linearity of CNNs. The conclusion of this study goes against the linearity hypothesis. The experimental studies of these values do not indicate any linear structure for CNNs. According to the modified linearity hypothesis, proposed in [42], CNNs are locally linear around the objects recognized by the model. The locality assumption is crucial here. Deep neural networks are non-linear in general and cannot be replaced completely by a linear classifier. However, the local linearity hypothesis claims that these models in a neighborhood of an instance can be approximated by a linear classifier. Additionally, the CNNs can be non-linear around those instances that are not recognized by the model.

The local linearity hypothesis implies that the decision boundaries around an instance can be approximated by a linear boundary. The geometric notion for characterizing the linearity of a surface is its curvature, as decision boundaries for binary classifiers are surfaces on higher dimension. The differential geometric notions of curvature are complex to characterize for DNNs. Therefore, in [19] the authors came up with an alternative, yet related definition of curvature. Consider the decision boundary for a binary classification problem<sup>4</sup> with the classifier  $f(\cdot)$  defined as

$$\mathcal{B} = \{\mathbf{x} : f(\mathbf{x}) = 0\},$$

and decision regions given by

$$\mathcal{R}_1 = \{\mathbf{x} : f(\mathbf{x}) > 0\} \quad \text{and} \quad \mathcal{R}_{-1} = \{\mathbf{x} : f(\mathbf{x}) < 0\}.$$

The curvature of the decision boundary  $\mathcal{B}$  with respect to  $\ell_q$ -norm is defined by

$$\kappa_q(\mathcal{B}) = \frac{1}{r_{\min}} \quad \text{where} \\ r_{\min} = \inf_{\mathbf{x} \in \mathcal{B}} \min_{i \in \{-1, 1\}} \sup_{\mathbf{x}_o \in \mathbb{R}^M} \{\|\mathbf{x}_o - \mathbf{x}\|_q : \mathbf{B}_q(\mathbf{x}_o, \|\mathbf{x}_o - \mathbf{x}\|_q) \subseteq \mathcal{R}_i\}$$

and  $B_q(\mathbf{x}, \epsilon)$  denotes the  $\ell_q$ -ball of radius  $\epsilon$  centered at  $\mathbf{x}$ . In other words  $r_{\min}$  is obtained by first finding at each point  $\mathbf{x}$ , on the decision boundary, the largest radius of  $\ell_q$ -balls that contain  $\mathbf{x}$  while being contained in  $\mathcal{R}_1$  and  $\mathcal{R}_{-1}$ . The radius is infinity for all  $q \geq 1$  when the decision boundary is flat, which means that the local curvature

---

<sup>4</sup>In this section, we focus mainly on binary classification examples assuming that the results can be extended without particular difficulty to multi-class classification problems.

is equal to zero at this point. The minimum of such radii for all  $\mathbf{x}$ , that is  $r_{\min}$ , points at the most curved portion of the surface. The global curvature of the surface  $\mathcal{B}$  is the inverse of  $r_{\min}$ . A linear classifier yields decision boundaries with zero curvature, and a small curvature surface might be completely flat in most of its points. It turns out that the  $\ell_q$ -curvature  $\kappa_q(\mathcal{B})$  determines the robustness against  $\ell_p$ -attacks with  $(p, q)$  as a dual pair.

It is based on this notion of curvature and for  $\ell_2$ -attacks that the authors in [19] compare the robustness of classifiers to random noise and adversarial noise and characterize it according to the curvature of decision boundaries.<sup>5</sup> The random noise is modeled as a random direction and the robustness against random noise at  $\mathbf{x}$ , denoted by  $\rho_M(\mathbf{x})$ , is defined by the minimum  $\ell_2$ -norm of a random vector required to change the label of  $\mathbf{x}$ . The adversarial robustness, denoted by  $\rho(\mathbf{x})$ , is the minimum  $\ell_2$ -norm of a perturbation particularly designed to change the label.

**Theorem 4** ([19, Theorem 2]) *Suppose that for a binary classifier the curvature  $\kappa_2(\mathcal{B})$  satisfies*

$$\kappa_2(\mathcal{B}) \leq \frac{0.2}{\zeta_2(\delta) M \rho(\mathbf{x})},$$

*then with probability at least  $1 - 4\delta$  it holds*

$$\begin{aligned} (1 - 0.625M\rho(\mathbf{x})\kappa_2(\mathcal{B})\zeta_2(\delta))\sqrt{M\zeta_1(\delta)} &\leq \frac{\rho_M(\mathbf{x})}{\rho(\mathbf{x})} \\ &\leq (1 + 2.25M\rho(\mathbf{x})\kappa_2(\mathcal{B})\zeta_2(\delta))\sqrt{M\zeta_2(\delta)}, \end{aligned}$$

*where*

$$\begin{aligned} \zeta_1(\delta) &= \left(1 + 2\sqrt{\ln(1/\delta)} + 2\ln(1/\delta)\right)^{-1}, \\ \zeta_2(\delta) &= \left(\max\left((1/e)\delta^2, 1 - \sqrt{2(1 - \delta^2)}\right)\right)^{-1}. \end{aligned}$$

The theorem implies that if the curvature of decision boundaries are small enough, the robustness of random noise is scaled with  $1/\sqrt{M}$  of the adversarial perturbation. In other words, in higher dimensions, classifiers with flat boundaries can be robust to random noise even if they are not robust to adversarial examples. Note that the curvature of non-smooth boundaries can be huge, rendering the above theorem non-informative. The curvature for multi-class classification, however, is characterized by the curvature of pairwise boundaries which does not include high curvature junctions. The intersection of these boundaries might have high curvature, but this issue does not matter in the above theorem where the pairwise boundaries are considered. Although only for  $\ell_2$ -attacks, this result is extended to  $\ell_p$ -attacks in [21], where the small curvature condition is replaced by a condition called locally approximately flat decision boundaries.

---

<sup>5</sup>They consider semi-random noise as well, however, we restrict ourselves to simple random noise.

The geometric properties of decision boundaries are further investigated in [46] for universal adversarial perturbations. Universal adversarial perturbations exist for both flat and curved decision boundaries. The essential to the existence of universal adversarial perturbations are shared directions along which the surface is positively curved. The above results compare random noise and adversarial perturbations. The result rely on low curvature assumptions of decision boundaries. The conclusion can be put with a flavor of blessing of dimensionality. That is, the locally flat classifiers are more robust to random noise particularly in higher dimensions.

The local linearity assumption is further refined in [16]. The flatness of decision boundaries can be violated in some directions, and nevertheless, the adversarial vulnerability persists. The authors show that the adversarial examples can exist if the boundaries are flat along most of the directions and highly curved only around few directions. This claim is additionally supported by [20, 55] where the curvature profile of deep networks are numerically characterized and is shown to be highly sparse, which implies that the boundaries are not flat overall but effectively only along most directions.

It is worth to finish the discussion around the linearity hypothesis by referring to a recent result that sheds some doubts and raises some questions about the role of linearity in adversarial robustness. Contrary to all above claims, the work [46] shows that the adversarial training, a powerful and consistent defense against adversarial attacks, leads to significant decrease in the curvature of loss functions. The connection runs in both directions as training DNNs with curvature regularization tends to improve the adversarial robustness. As long as the curvature of loss functions affects the curvature of decision boundaries, the result stands out as a strong counter-argument for linearity hypothesis and opens new challenges for it.

### 3.2 Boundary Tilting and Other Explanations

The linear hypothesis is not the only available theory. Other theories attribute adversarial robustness to other features of classifiers. A simple intuition already emerged from our discussion of linear classifiers. The  $\ell_\infty$ -attacks for linear classifiers with unit-norm parameters  $\mathbf{w}$  generated a perturbation equal to  $-\epsilon \|\mathbf{w}\|_1$ . Therefore, among all unit-norm  $\mathbf{w}$ 's, the most robust classifiers are those with smallest  $\ell_1$ -norm, which are also the sparsest possible vectors.

For binary classification problems, the authors in [25] showed theoretically that the adversarial robustness decreases when  $\ell_1$ -norm of  $\mathbf{w}$  increases. We introduce some definitions before stating the theorem. Suppose that the instances and labels  $(\mathbf{x}, y)$  follow a distribution  $P_{\mathbf{x}, y}$ . The adversarial robustness for this probabilistic model is defined as

$$\rho_\infty = P_{\mathbf{x}, y}[y \neq \text{sign}(\mathbf{w}^\top \mathbf{x}_{\text{adv}})],$$

where  $\mathbf{x}_{\text{adv}}$  is the  $\ell_\infty$ -perturbed instance and given by  $\mathbf{x}_{\text{adv}} = \mathbf{x} - \epsilon y \text{sign}(\mathbf{w})$ . Let us define  $\mu_k$  for  $k \in \{1, -1\}$  as follows

$$\mu_k = \mathbb{E}(\mathbf{x}|y=k, \text{sign}(\mathbf{w}^\top \mathbf{x})=k).$$

We can now state the theorem.

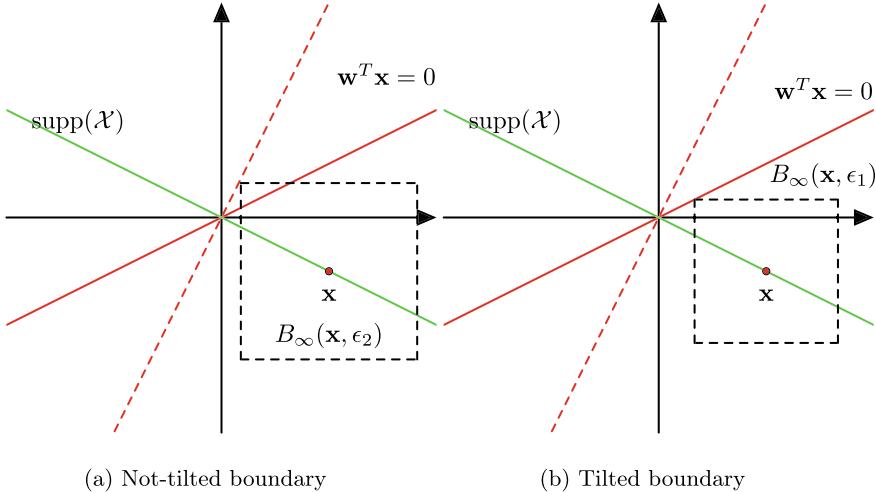
**Theorem 5** ([25, Theorem 3.1]) *For a binary classification problem with uniformly distributed labels, if the accuracy of a classifier is given by  $t$  then the adversarial robustness  $\rho_\infty$  against  $\epsilon$  bounded  $\ell_\infty$  attacks is given by*

$$\rho_\infty \leq \frac{t\mathbf{w}^\top(\boldsymbol{\mu}_{+1} - \boldsymbol{\mu}_{-1})}{2\epsilon\|\mathbf{w}\|_1}.$$

The denominator of the bound on the right hand side contains the  $\ell_1$ -norm of  $\mathbf{w}$ . Therefore, as the authors in [25] maintain, among those linear classifiers with a similar discriminatory capability, those with the smallest  $\ell_1$ -norm perform better under  $\ell_\infty$ -attacks. The small  $\ell_1$ -norm implies a larger  $\rho_\infty$ , which means better robustness.

The theorem, however, provides only an upper bound, and, although it can characterize the negative effect of large  $\ell_1$ -norm on robustness, it cannot necessarily guarantee that the small  $\ell_1$ -norm promotes robustness nor that small  $\ell_1$ -norms necessarily lead to sparse  $\mathbf{w}$ . The claim, however, seems to hold, as experimental findings seem to support the idea that the sparsity of weights promote adversarial robustness. Besides, since the difference  $\boldsymbol{\mu}_{+1} - \boldsymbol{\mu}_{-1}$  is independent of the norm of  $\mathbf{w}$ , the inner product  $\mathbf{w}^\top(\boldsymbol{\mu}_{+1} - \boldsymbol{\mu}_{-1})$  scales with the norm of  $\mathbf{w}$ . In this light, another reading of this theorem suggests that among all unit  $\ell_1$ -norm  $\mathbf{w}$ 's, the one with smallest  $\mathbf{w}^\top(\boldsymbol{\mu}_{+1} - \boldsymbol{\mu}_{-1})$  restricts robustness the least. To summarize the theorem, a first step toward robustness of linear classifiers is to find the smallest  $\ell_1$ -norm  $\mathbf{w}$  for which the inner product  $\mathbf{w}^\top(\boldsymbol{\mu}_{+1} - \boldsymbol{\mu}_{-1})$  is high enough.

Another explanation of adversarial examples, introduced in [71], starts from the assumption that the data lies on a low-dimensional manifold in higher dimensional space, and many classifiers exist with similar accuracy. This is shown in Fig. 5 using a simple example of linear manifolds and linear classifiers. We assume that the data lies on a linear subspace and the dashed line represents the boundary of an optimal Bayes linear classifier for the data distribution with zero error. However, the rotated versions of this linear classifier, for example the one with the solid line as the boundary, yield the same accuracy. The main difference between these classifiers is their robustness to adversarial examples. If the linear boundary is tilted so that it lies close to the data subspace, the smaller  $\ell_\infty$ -norm perturbation can fool the classifier. This can be seen in Fig. 5 as the  $\ell_\infty$ -ball touching the tilted classifier is smaller than the original not-tilted classifier. This is known under boundary tilted hypothesis. Under this hypothesis, the adversarial vulnerability of classifiers arises from the tilted classification boundary close to the data manifold. A further exploration of the linear classifier example can reveal that some of the tilted boundaries can indeed improve the robustness.



**Fig. 5** Adversarial robustness of tilted boundaries: **a** the dashed line is the ground truth linear classifier for the data supported on  $\mathcal{X}$ , **b** the solid line, a tilted boundary, yields the same risk as the ground truth but is fooled with smaller  $\ell_\infty$ -perturbations

### 3.3 Feature Selection and No Free Lunch Theorems for Adversarial Robustness

Many classifier functions can be clearly decomposed into feature extraction and classification parts. In [74], adversarial robustness is shown to be affected by the feature selection part of the model. The results of [74] rely on the assumption that there is an oracle classifier function  $g(\mathbf{x})$  that generates the ground truth labels. For image classification problems, it is simply the human eye. Classifiers  $f(\cdot)$ , in particular  $g(\cdot)$ , are decomposed into a feature extraction part  $e_f(\cdot)$  and a classifier part  $c_f(\cdot)$ . The feature spaces of a classifier is the image of the domain set  $\mathcal{X}$  under the feature extraction  $e_f(\cdot)$ . Feature spaces are assumed to be metric spaces. Denote the oracle feature space by  $(\mathcal{X}_g, d_g)$  where  $d_g$  is the respective metric, and  $(\mathcal{X}_f, d_f)$  similarly for a classifier  $f(\cdot)$ .

Adversarial perturbations do not change the oracle decision, i.e.,  $g(\mathbf{x}) = g(\mathbf{x} + \boldsymbol{\eta})$  nor the feature extraction:

$$d_g(e_g(\mathbf{x}), e_g(\mathbf{x} + \boldsymbol{\eta})) < \delta.$$

However, the classifier is fooled ( $f(\mathbf{x}) \neq f(\mathbf{x} + \boldsymbol{\eta})$ ). A classifier is called  $(\epsilon, \delta)$ -robust if for all  $\mathbf{x}, \mathbf{y} \in \mathcal{X}$  for which  $g(\mathbf{x}) = g(\mathbf{y})$  and  $d_g(e_g(\mathbf{x}), e_g(\mathbf{y})) < \epsilon$  then with probability at least  $1 - \delta$  it holds that  $f(\mathbf{x}) \neq f(\mathbf{y})$ .

**Theorem 6** ([74, Theorem 3.2–3.4]) *Let a classifier  $f(\cdot)$  be continuous almost everywhere and  $g(\cdot)$  be the oracle classifier. The classifier  $f(\cdot)$  is  $(\epsilon, \delta)$ -robust to adversarial examples if and only if the topology of the feature space  $(\mathcal{X}_f, d_f)$  is finer than the topology of the oracle feature space  $(\mathcal{X}_g, d_g)$ .*

As a direct corollary of the above theorem, when the two features spaces are Euclidean spaces of dimension  $n_g$  and  $n_f$ , then the classifier  $h(\cdot)$  is robust if and only if  $n_f < n_g$ . The above theorem implies first that the selection of features and feature spaces is crucial for adversarial robustness. Although the assumption of an oracle function and a unique suitable feature space can be contested, the theorem applies to any two classifiers and states that if a perturbation does not fool  $g(\cdot)$  then it does not fool  $f(\cdot)$ . So among the classifiers those with feature spaces of finer topology, or lower dimension in Euclidean spaces, are favored for adversarial robustness.

The importance of selecting proper features is addressed in other works such as [17, 18]. A toy example is used in [17, 18] to show that linear classifiers are unable to use more robust features of an image for adversarial robust classification, unlike quadratic classifiers that are more robust in that example. For  $\ell_2$ -attacks, the authors point out that the adversarial robustness is directly related to the so-called distinguishability measure of classes and the risk of the classifier. The distinguishability measure can be seen as low flexibility of classifiers in general compared to the difficulty of the classification task. We state a simplified version of their theorem for linear classifiers using the definition of  $\rho(\mathbf{x})$  from Theorem 4.

**Theorem 7** ([17, Theorem 4.1]) *For a binary classification task with uniformly distributed labels and  $\|\mathbf{x}\|_2 \leq B$  a.e., the adversarial robustness of a linear classifier  $\text{sign}(\mathbf{w}^\top \mathbf{x})$  with accuracy  $t$  satisfies*

$$\mathbb{E}(\rho(\mathbf{x})) \leq \frac{1}{2} \|\mathbb{E}_{P_{\mathbf{x}|y=+1}}(\mathbf{x}) - \mathbb{E}_{P_{\mathbf{x}|y=-1}}(\mathbf{x})\|_2 + 2Bt.$$

The distinguishability measure  $\|\mathbb{E}_{P_{\mathbf{x}|y=+1}}(\mathbf{x}) - \mathbb{E}_{P_{\mathbf{x}|y=-1}}(\mathbf{x})\|_2$  is a feature of classification problem and not dependent on the classifier. However, an unexpected conclusion of the theorem is that if the classification task is difficult, in that the distinguishability measure is small, the risk of the classifier becomes dominant in the upper bound and inversely related with the robustness. Therefore, low risk classifiers have less adversarial robustness for difficult classification tasks.

The inverse connection of risk and robustness is further explored in [73] through a binary classification example. An instance of data is given by  $\mathbf{x} = (x_1, \dots, x_n, x_{n+1})^\top$  and it is related to its label  $y$  randomly as follows. The first entry is a Bernoulli random variable with  $P(x_1 = y) = p$  and the other entries,  $x_i$ , are normal distributed random variables with mean value  $\xi y$  and unit variance. A linear classifier with  $\mathbf{w} = (0, 1/n, \dots, 1/n)$  can be shown to achieve more than 0, 99 accuracy if  $\xi = \Theta(1/\sqrt{M})$ . However, this classifier can achieve an adversarial accuracy at most 0.01 under the  $\ell_\infty$ -attack with  $\epsilon = 2\xi$ . However, if one uses only the first feature  $x_1$  for the classification both standard and adversarial accuracies are 0.7. The data consists of, on the one hand, robust features with less accuracy and, on the other hand, informative and non-robust features. We might ask whether this tension can be circumvented using a smart combination of features so that the adversarial robustness does not come at the price of accuracy. The authors of [73] answer negatively by stating a no free-lunch theorem for adversarial robustness.

**Theorem 8** ([73, Theorem 2.1]) Any classifier with standard accuracy at least  $1 - \delta$  on the above problem cannot achieve adversarial accuracy more than  $\frac{1-p}{p}\delta$  against  $\ell_\infty$ -bounded perturbations with  $\|\eta\|_\infty \geq 2\xi$ .

The importance of feature selection for adversarial robustness is highlighted already in [73]. A similar result is obtained in [15] for a class of data distributions satisfying  $W_2$ -Talagrand transportation-cost inequality. The condition is intuitively related to the curvature of decision regions and matches the previously mentioned intuition that low curvature decision boundaries entail adversarial vulnerability. From another perspective,  $\epsilon$ -bounded  $\ell_p$ -adversarial attacks manage to alter the label of instances that are included in the  $\epsilon \ell_p$ -boundary of decision regions. The adversarial problem so formulated, naturally, can be cast as the study of blowing-up property of decision regions—a problem well studied by concentration results and isoperimetric inequalities. A similar approach is followed in [14, 22, 44] for instances following Gaussian distribution and uniform-distribution over hypercubes.

### 3.4 Generalization Bounds for Adversarial Examples

The no-free-lunch-theorem states that adversarial robustness of machine learning algorithms does not align in general with their risk. Adversarial training of DNNs, indeed, confirm the same point that adversarial robustness is obtained at the cost of degraded generalization. According to an example in [43], the adversarial accuracy of 96% for ResNet dataset trained on CIFAR-10 comes with a test accuracy of 47%. In [67], the authors attribute this trade-off to the definition of adversarial robustness. They propose another definition of adversarial robustness for which no trade-off is observed between adversarial robustness and accuracy.

From these indications, therefore, emerge questions regarding generalization properties of adversarially robust algorithms. We summarize some of the progresses in that direction that use statistical learning theory as the framework for studying generalization properties of learning algorithms. An interested reader can refer to the excellent manuals [2, 64] for introduction to fundamental notions of statistical learning theory.

Statistical learning theory explains generalization properties of many classical learning algorithms using notions like VC-dimension, Rademacher complexity and uniform convergence. A similar approach, however, cannot be directly applied to understanding generalization for neural networks. The large number of parameters in DNNs renders many of these bounds ultimately useless as these models are capable of fitting arbitrary large number of random instances with random labeling [79]. A solution is to use a suitable normalization. It has been shown that the margin based normalization can be used to obtain tight generalization bounds by Rademacher complexity or PAC-Bayesian methods that match experimental results [3, 10, 23, 49].

We already discussed some works relating adversarial robustness and accuracy, all of them derived for a class of data distributions. In this section, we focus on sample complexity bounds for adversarially robust generalization and see whether the available bounds attest to difficulty of training adversarially robust and yet accurate models. A first indication in this direction can be traced to [63] where it is shown that for Gaussian model of data, the sample complexity of robust learning for  $M$ -dimensional data is  $\Theta(\sqrt{M})$  times larger than standard learning, hence, more difficulty of the former. The gap is information theoretic. PAC-learning for the adversarial setting is an open problem, although some bounds exist for binary linear classifiers obtained in terms of VC-dimension [13]. The former result, however, show no negative effect of robust training on generalization, counter to the above intuition. A generalization bound is also obtained in [6] when the set of adversarial perturbations is finite. The sample complexity of binary classification depends  $k \log(k) \text{VC}(\mathcal{H})$  where  $\text{VC}(\mathcal{H})$  is the VC-dimension of the hypothesis class  $\mathcal{H}$  and  $k$  is the number of different adversarial perturbations. The result is not directly applicable to standard adversarial attacks where the set of possible perturbations is not finite, however, it points to the larger sample complexity of robust learning.

There is a difficulty with VC-dimension bounds when they are applied to DNNs. As it is explained in [77], VC-dimension bounds depend usually on the number of model parameters, which is very large for DNNs. The corresponding sample complexity bound becomes unreasonably large beyond typical available datasets. Rademacher complexity bounds, on the other hand, depend mostly on inherently smaller quantities like the norm of weight matrices and, therefore, are more appropriate for establishing generalization property of DNNs.

Rademacher complexity bounds for adversarial robustness is obtained in [31, 77]. They use different techniques for deriving their bounds and have different scope of applicability. Nevertheless, both works contain Rademacher complexity bounds for binary and multi-class classification and are applicable to neural networks. Both works use surrogate adversarial loss. In particular, the authors in [77] build on semidefinite programming (SDP) relaxation techniques of [56]. We do not expand on the technical results and, instead, state qualitatively some implications of these results.

As we discussed above shortly, Rademacher complexity bounds depend mostly on the norm of weight matrices. However, the lower bound on Rademacher complexity of neural networks for robust training in [77] has additional dependence on the dimension for the  $\ell_\infty$ -attacks. The dependence disappears only if the weight matrix of the first layer has bounded  $\ell_1$ -norm. In absence of this assumption, this bound confirms the hypothesis that robust training is more difficult than standard training. The technique employed in [31], however, yields upper bounds in which the effect of adversarial perturbations appears as an additive term in the generalization bound. The authors, therefore, conclude that it should not be impossible to obtain both high adversarial robustness and high accuracy. Although a final verdict seems to be far reaching at the moment, new regularization techniques arises from these generalization bounds that can be used during training for robust learning with high accuracy.

## 4 Defenses Against Adversarial Attacks

There exist several types of defenses against adversarial examples, as well as subsequent methods for bypassing them. It is difficult to point out, at the time of writing this chapter, a consensus on the effective defense against adversarial examples with the possible exception of adversarial training. For instance, the authors in [11] proposed three attacks to bypass defensive distillation of the adversarial perturbations [54]. Moreover, the attacks from [5], bypassed 7 out of 9 non-certified defenses of ICLR 2018 that claimed to be white-box secure. Adversarial training, however, adds adversarial examples to the training set and is the most commonly accepted defense against adversarial attacks. In what follows, we discuss some difficulties of adversarial training as well as those methods that try to promote robustness merely through regularization techniques.

### 4.1 *Obfuscated Gradients and Adversarial Training*

The rise of adversarial perturbations in computer vision has motivated further research on defending against such perturbations. To that end several defenses against adversarial examples, such as [38, 60, 65], have been designed. Since many adversarial attacks make use of the classifier’s gradient with respect to some objective function, as in Algorithm 1, initial works on adversarial defenses rely on distorting and hiding the information about that gradient. In [5], these techniques were said to *obfuscate* the gradient. More precisely, obfuscating the gradient may be done in either of the following manners.

- **Shattered Gradients** appear when the defense mechanism is not differentiable, numerically unstable, or intentionally has misleading gradients.
- **Stochastic Gradients** occur when the defense method is based on introducing randomness into the prediction. Such randomness is added to prevent the attacker from estimating the gradients.
- **Exploding Gradients** happen when the defense algorithm consists on recursive evaluations of the DNN function. In other words, the output of one DNN evaluation is the input of the next one. This type of computation implicitly transforms the original DNN into a extremely deep neural network, which may lead gradients to explode (or vanish) during inference.

Despite the apparent success of this type of defenses, in [5] it is shown that such mechanisms give a false sense of security. The main reason behind this phenomena is that obfuscating the gradients of model does not necessarily increases its robustness against adversarial perturbations, instead it prevents specific algorithms to find them. In other words, for such models adversarial perturbations still exist; they are just harder to find using certain methods. Therefore, although a model with obfuscated gradients can be robust against a specific adversarial attack, it may still be vulnerable

to others. Using this idea, the authors in [5] provide the following conditions to identify models that exhibit this problem.

- **One-step attacks perform better than iterative attacks.**
- **Black-box attacks perform better than white-box attacks.**
- **Attacks with large  $\epsilon$  do not reach 100% fooling ratio.**
- **Random sampling finds adversarial examples, while adversarial attacks don't.**

If a model satisfies one of the above conditions, it suffers from the obfuscated gradient problem. Using these guidelines, the authors identified that 7 out of 9 defenses accepted to ICLR 2018, that were deemed to be white-box secure, suffered from this issue. In addition, the authors fooled those defenses using customized attacks.

So far, the most successful defenses against adversarial attacks consist of adding adversarial examples to the training set. This is known as adversarial training. Initial attempts to perform adversarial training using the FGSM proved to suffer from obfuscated gradients. This occurs since a DNN trained solely with the FGSM learns to shatter the gradients that are in a close vicinity to the data samples, such that the gradients used for the FGSM point into misleading directions. While this process may mislead the FGSM it is still vulnerable to other perturbations, for instance black-box attacks. To overcome this issue the dithering mechanism proposed for the PGD attack is employed in [43], along with large  $\epsilon$  values<sup>6</sup> during adversarial training. This approach provided diverse sets of random adversarial examples, which prevented DNNs from obtaining low fooling ratios by shattering the gradients around the data samples. In other words, the randomness of the starting point in the PGD attack prevents the model from overfitting the perturbations.

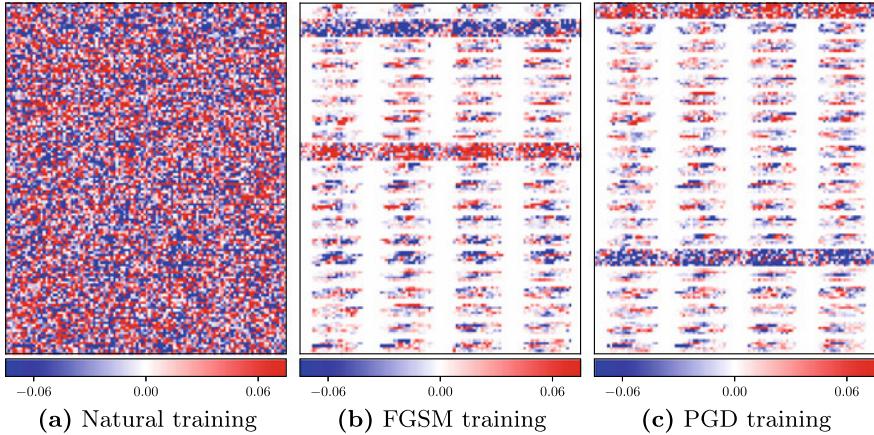
From these initial findings it is concluded that *diversity* in the adversarial examples used for training is necessary in order to prevent DNNs for overfitting to specific types of perturbations. To that end, in [72] the authors include black-box perturbations into the training set. This is carried out using substitute models, as well as ensembles of these models, in the objective function of white-box attacks (such as PGD) as described in Sect. 2.2.

## 4.2 Robust Regularization

Despite the success of adversarial training on promoting robustness, these methods either suffer from obfuscated gradients (e.g., when the FGSM is employed) or are deemed to be computationally expensive, since iterative methods require several evaluations of the DNN function to compute a single adversarial example. In [43] it is observed that adversarial training induces sparsity on the weights of the first convolutional filters of CNNs trained with the MNIST dataset. Similarly, the authors in [36] observe that adversarial training induces low-rank structures in the

---

<sup>6</sup>In that work, the  $\ell_\infty$ -constraint  $\|\boldsymbol{\eta}\|_\infty \leq \epsilon = 0.3$  is employed to train models where the input values were between 0 and 1.



**Fig. 6** Reshaped input weight matrix  $\mathbf{W}^1 \in \mathbb{R}^{20 \times 784}$  of a DNN, from [36], after natural training as well as adversarial training with  $\epsilon = 0.05$ . A simultaneously low-rank and sparse structure is observed in the weights after adversarial training

weight matrices of the DNN, as well sparsity. As an example the authors provide a visualization of the weights in the first layer of a DNN, shown in Fig. 6, where the simultaneously low-rank and sparse structure of such weights is clearly visible. This is additionally confirmed by looking at the mutual information between the input and the layers. Information theoretically, increasing adversarial robustness coincides with decreasing the mutual information that indicates more compression of the input in hidden layers. These results serve as motivation for aiming research towards finding the key properties that lead to robustness of DNNs. The idea is to propose a metric for robustness and promote it during training. A common technique for promoting specific properties during training is to add a penalty term in the loss function, known as regularization term, that penalizes undesired properties of the classifier function. Here are some examples of robust regularization.

- **Sparsity:** In [25, 75] the authors argue that sparsity of the weight matrices of a DNN promotes robustness against adversarial examples. They propose to add a regularization term with the sum of the  $\ell_1$ -norm of the weight matrices involved, which is known to promote approximately sparse solutions. In addition, the authors make use of *pruning*<sup>7</sup> to impose arbitrary sparsity levels.
- **Low-Rankness:** In [36, 61] it is observed that adversarial training induces low-rank structures on the weight matrices of DNNs. Motivated by this phenomena, low-rank regularization techniques are proposed. In [61] the authors explicitly

<sup>7</sup>Pruning consists of setting to zero smallest weights (in absolute value) of the a given weight matrix, thus enforcing a certain level of sparsity. The amount of weights to be set to zero is arbitrarily chosen. Usually pruning requires an extra phase of retraining (fine-tunning of the remaining non-zero weights) to compensate for the performance degradation caused by the initial manipulation of the weights.

constrain the rank of weight matrices in the optimization algorithm used for training. On the other hand in [36] the nuclear norm of the weight matrices is employed as a regularization term in the training loss. The nuclear norm of a matrix can be written as the  $\ell_1$ -norm of its vector of singular values, thus using it as a regularization term promotes sparsity in that vector of singular values (i.e., low-rankness).

- **Norm of the network’s Jacobian:** In [50], the authors aim at minimizing the  $\ell_2$ -norm of the output perturbation, that is  $\|f(\mathbf{x}) - f(\mathbf{x} + \boldsymbol{\eta})\|_2$ . Assuming  $\|\boldsymbol{\eta}\|_2 \leq \epsilon$ , upper-bounding an approximate of this functional yields

$$\|f(\mathbf{x}) - f(\mathbf{x} + \boldsymbol{\eta})\|_2 \approx \|\mathbf{J}_f(\mathbf{x})\boldsymbol{\eta}\|_2 \leq \epsilon \|\mathbf{J}_f(\mathbf{x})\|_{\text{F}}.$$

Motivated by this result, the authors propose using the Frobenius norm of the Jacobian  $\|\mathbf{J}_f(\mathbf{x})\|_{\text{F}}$  as regularization term to promote robustness. The Frobenius norm is an upper bound on the  $\ell_2$ -norm of the output perturbation. If it is limited during training by proper regularization, it can restrict the  $\ell_2$ -perturbations.

- **Curvature:** In Sect. 3.1 it is argued that low curvature in the decision boundaries, as well as in the loss function, are desired properties for robustness. Motivated by that discussion, the authors of [48] proposed penalizing solutions with high curvature of the loss function around the training data.

## 5 Future Directions

Adversarial examples appear as a potential obstacle for widespread employment of DNNs particularly in safety critical applications. An ultimate solution, yet, seems to be out of reach even for a simple task of MNIST image classification. Adversarial training is the best known defense in this situation that comes with two additional problems. It is computationally costly and degrades the generalization of DNNs. Future works can see whether the latter problem can be solved by different training techniques, or the generalization degradation cost is to be paid inevitably for more robustness. There is no consensus on the nature of adversarial examples and which features of classifiers play the central role in adversarial robustness. There are many indications that occasionally align with experimental results. Recent statistical learning theory approaches, however, provide a promising path to address generalization and robustness simultaneously. The ultimate goal of an adequate account of adversarial robustness, although not attained so far, constitutes an exciting field of research in coming years.

**Acknowledgements** The authors would like to thank the reviewers for their fruitful feedbacks.

## References

1. Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: a survey. *IEEE Access* **6**, 14410–14430 (2018)
2. Anthony, M., Bartlett, P.L.: *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, Cambridge (2009)
3. Arora, S., Ge, R., Neyshabur, B., Zhang, Y.: Stronger generalization bounds for deep nets via a compression approach. In: International Conference on Machine Learning, pp. 254–263 (2018)
4. Athalye, A., Carlini, N.: On the Robustness of the CVPR 2018 white-box adversarial example defenses (Apr 2018). [arXiv:1804.03286](https://arxiv.org/abs/1804.03286)
5. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. In: International Conference on Machine Learning (2018)
6. Attias, I., Kontorovich, A., Mansour, Y.: Improved generalization bounds for robust learning. In: Algorithmic Learning Theory, pp. 162–183 (2019)
7. Balda, E.R., Behboodi, A., Mathar, R.: On generation of adversarial examples using convex programming. In: 52-th Asilomar Conference on Signals Systems, and Computers, pp. 1–6. Pacific Grove, California, USA (Oct 2018)
8. Balda, E.R., Behboodi, A., Mathar, R.: Perturbation analysis of learning algorithms: generation of adversarial examples from classification to regression. *IEEE Trans. Signal Process.* (2018)
9. Barreno, M., Nelson, B., Joseph, A.D., Tygar, J.D.: The security of machine learning. *Mach Learn* **81**(2), 121–148 (2010)
10. Bartlett, P.L., Foster, D.J., Telgarsky, M.J.: Spectrally-normalized margin bounds for neural networks. In: Advances in Neural Information Processing Systems, vol. 30, pp. 6240–6249. Curran Associates, Inc. (2017)
11. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE (2017)
12. Cisse, M., Adi, Y., Neverova, N., Keshet, J.: Houdini: fooling deep structured prediction models (2017). [arXiv:1707.05373](https://arxiv.org/abs/1707.05373)
13. Cullina, D., Bhagoji, A.N., Mittal, P.: PAC-learning in the presence of adversaries. In: Advances in Neural Information Processing Systems, vol. 31, pp. 228–239. Curran Associates, Inc. (2018)
14. Diochnos, D., Mahloujifar, S., Mahmoodi, M.: Adversarial risk and robustness: general definitions and implications for the uniform distribution. In: Advances in Neural Information Processing Systems, pp. 10359–10368 (2018)
15. Dohmatob, E.: Limitations of adversarial robustness: strong No Free Lunch Theorem (Oct 2018). [arXiv:1810.04065](https://arxiv.org/abs/1810.04065) [cs, stat]
16. Fawzi, A., Moosavi-Dezfooli, S.M., Frossard, P.: The robustness of deep networks: a geometrical perspective. *IEEE Signal Process. Mag.* **34**(6), 50–62 (2017)
17. Fawzi, A., Fawzi, O., Frossard, P.: Fundamental limits on adversarial robustness. In: Proceedings of ICML, Workshop on Deep Learning (2015)
18. Fawzi, A., Fawzi, O., Frossard, P.: Analysis of classifiers' robustness to adversarial perturbations. *Mach. Learn.* **107**(3), 481–508 (2018)
19. Fawzi, A., Moosavi-Dezfooli, S.M., Frossard, P.: Robustness of classifiers: from adversarial to random noise. In: Advances in Neural Information Processing Systems, vol. 29, pp. 1632–1640. Curran Associates, Inc. (2016)
20. Fawzi, A., Moosavi-Dezfooli, S.M., Frossard, P., Soatto, S.: Classification regions of deep neural networks (May 2017). [arXiv:1705.09552](https://arxiv.org/abs/1705.09552)
21. Franceschi, J.Y., Fawzi, A., Fawzi, O.: Robustness of classifiers to uniform  $\ell_p$  and Gaussian noise. In: 21st International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 84, p. 9. Lanzarote, Spain (2018)
22. Gilmer, J., Metz, L., Faghri, F., Schoenholz, S., Raghu, M., Wattenberg, M., Goodfellow, I.: Adversarial spheres. In: ICLR 2018-Workshop Track (2018)

23. Golowich, N., Rakhlin, A., Shamir, O.: Size-independent sample complexity of neural networks. In: Bubeck, S., Perchet, V., Rigollet, P. (eds.) Proceedings of the 31st Conference On Learning Theory. Proceedings of Machine Learning Research, vol. 75, pp. 297–299. PMLR (Jul 2018)
24. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (Dec 2014)
25. Guo, Y., Zhang, C., Zhang, C., Chen, Y.: Sparse DNNs with improved adversarial robustness. In: Advances in Neural Information Processing Systems, vol. 31, pp. 240–249. Curran Associates, Inc. (2018)
26. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (Jun 2016)
27. Hein, M., Andriushchenko, M.: Formal guarantees on the robustness of a classifier against adversarial manipulation. In: NIPS (2017)
28. Hendrik Metzen, J., Chaithanya Kumar, M., Brox, T., Fischer, V.: Universal adversarial perturbations against semantic image segmentation. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2755–2764 (2017)
29. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
30. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017)
31. Khim, J., Loh, P.L.: Adversarial risk bounds via function transformation (Oct 2018). [arXiv:1810.09519](https://arxiv.org/abs/1810.09519)
32. Khrulkov, V., Oseledets, I.V.: Art of singular vectors and universal adversarial perturbations. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8562–8570 (2018)
33. Kos, J., Fischer, I., Song, D.: Adversarial examples for generative models. In: 2018 IEEE Security and Privacy Workshops (SPW), pp. 36–42 (May 2018)
34. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
35. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world (2016). [arXiv:1607.02533](https://arxiv.org/abs/1607.02533)
36. Langenberg, P., Balda, E.R., Behboodi, A., Mathar, R.: On the effect of low-rank weights on adversarial robustness of neural networks (2019). [arXiv:1901.10371](https://arxiv.org/abs/1901.10371)
37. LeCun, Y., Haffner, P., Bottou, L., Bengio, Y.: Object recognition with gradient-based learning. In: Shape, Contour and Grouping in Computer Vision, pp. 319–345. Springer (1999)
38. Liao, F., Liang, M., Dong, Y., Pang, T., Hu, X., Zhu, J.: Defense against adversarial attacks using high-level representation guided denoiser. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
39. Lin, M., Chen, Q., Yan, S.: Network in network (2013). [arXiv:1312.4400](https://arxiv.org/abs/1312.4400)
40. Lin, Y.C., Hong, Z.W., Liao, Y.H., Shih, M.L., Liu, M.Y., Sun, M.: Tactics of adversarial attack on deep reinforcement learning agents. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 3756–3762. IJCAI’17, AAAI Press, Melbourne, Australia (2017)
41. Liu, Y., Chen, X., Liu, C., Song, D.: Delving into transferable adversarial examples and black-box attacks. In: ICLR 2017 (2017)
42. Luo, Y., Boix, X., Roig, G., Poggio, T., Zhao, Q.: Foveation-based mechanisms alleviate adversarial examples (Nov 2015). [arXiv:1511.06292](https://arxiv.org/abs/1511.06292)
43. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018)

44. Mahloujifar, S., Diochnos, D.I., Mahmoody, M.: The curse of concentration in robust learning: evasion and poisoning attacks from concentration of measure. In: Thirty-Third AAAI Conference on Artificial Intelligence (AAAI) (2019)
45. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1765–1773 (2017)
46. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P., Soatto, S.: Robustness of classifiers to universal perturbations: a geometric perspective. In: International Conference on Learning Representations (2018)
47. Moosavi Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
48. Moosavi-Dezfooli, S.M., Fawzi, A., Uesato, J., Frossard, P.: Robustness via curvature regularization, and vice versa (Nov 2018). [arXiv:1811.09716](https://arxiv.org/abs/1811.09716) [cs, stat]
49. Neyshabur, B., Bhojanapalli, S., Srebro, N.: A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. In: International Conference on Learning Representations (2018)
50. Novak, R., Bahri, Y., Abolafia, D.A., Pennington, J., Sohl-Dickstein, J.: Sensitivity and generalization in neural networks: an empirical study. In: International Conference on Learning Representations (2018)
51. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 506–519. ACM (2017)
52. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016)
53. Papernot, N., McDaniel, P., Swami, A., Harang, R.: Crafting adversarial input sequences for recurrent neural networks. In: Military Communications Conference, MILCOM 2016–2016 IEEE, pp. 49–54. IEEE (2016)
54. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 582–597. IEEE (2016)
55. Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., Ganguli, S.: Exponential expressivity in deep neural networks through transient chaos. In: Advances in Neural Information Processing Systems, vol. 29, pp. 3360–3368. Curran Associates, Inc. (2016)
56. Raghunathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: International Conference on Learning Representations (2018)
57. Rao, N., Recht, B., Nowak, R.: Universal measurement bounds for structured sparse signal recovery. In: Artificial Intelligence and Statistics, pp. 942–950 (Mar 2012)
58. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2017)
59. Sabour, S., Cao, Y., Faghri, F., Fleet, D.J.: Adversarial manipulation of deep representations. In: ICLR 2016 (2016)
60. Samangouei, P., Kabkab, M., Chellappa, R.: Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In: International Conference on Learning Representations (2018)
61. Sanyal, A., Kanade, V., Torr, P.H.S.: Intriguing properties of learned representations (2018)
62. Sarkar, S., Bansal, A., Mahbub, U., Chellappa, R.: Upset and angri: breaking high performance image classifiers (2017). [arXiv:1707.01159](https://arxiv.org/abs/1707.01159)
63. Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., Madry, A.: Adversarially robust generalization requires more data. In: Advances in Neural Information Processing Systems, pp. 5014–5026 (2018)
64. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, New York, NY, USA (2014)

65. Song, Y., Kim, T., Nowozin, S., Ermon, S., Kushman, N.: Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In: International Conference on Learning Representations (2018)
66. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. (2019)
67. Suggala, A.S., Prasad, A., Nagarajan, V., Ravikumar, P.: Revisiting adversarial risk (Jun 2018). [arXiv:1806.02924](https://arxiv.org/abs/1806.02924) [cs, stat]
68. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
69. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)
70. Tabacof, P., Tavares, J., Valle, E.: Adversarial images for variational autoencoders (2016). [arXiv:1612.00155](https://arxiv.org/abs/1612.00155)
71. Tanay, T., Griffin, L.: A boundary tilting persepective on the phenomenon of adversarial examples (Aug 2016). [arXiv:1608.07690](https://arxiv.org/abs/1608.07690) [cs, stat]
72. Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P.: Ensemble adversarial training: attacks and defenses. In: International Conference on Learning Representations (2018)
73. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness may be at odds with accuracy. In: International Conference on Learning Representations (2019)
74. Wang, B., Gao, J., Qi, Y.: A theoretical framework for robustness of (Deep) classifiers against adversarial examples. In: International Conference on Learning Representations (2017)
75. Wang, L., Ding, G.W., Huang, R., Cao, Y., Lui, Y.C.: Adversarial robustness of pruned neural networks (2018). <https://openreview.net/forum?id=SJGrAisIz>
76. Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., Yuille, A.: Adversarial examples for semantic segmentation and object detection. In: International Conference on Computer Vision. IEEE (2017)
77. Yin, D., Ramchandran, K., Bartlett, P.: Rademacher complexity for adversarially robust generalization (Oct 2018). [arXiv:1810.11914](https://arxiv.org/abs/1810.11914) [cs, stat]
78. Yuan, X., He, P., Zhu, Q., Li, X.: Adversarial examples: attacks and defenses for deep learning. IEEE Trans. Neural Netw. Learn. Syst. 1–20 (2019)
79. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: ICLR 2018 (2017)

# Representation Learning in Power Time Series Forecasting



Janosch Henze, Jens Schreiber and Bernhard Sick

**Abstract** Renewable energy resources have become a fundamental part of the electrical power supply in many countries. In Germany, renewable energy resources contribute up to 29% to the energy mix. However, the challenges that arise with the integration of those variable energy resources are various. Some of these tasks are short-term and long-term power generation forecasts, load forecasts, integration of multiple numerical weather prediction (NWP) models, simultaneous power forecasts for many renewable farms and areas, scenario generation for renewable power generation, and the list goes on. All these tasks vary in difficulty depending on the representation of input features. As an example, consider formulas that express laws of physics and allow cause and effect of otherwise complex problems to be calculated. Similar to the expressiveness of such formulas, deep learning provides a framework to represent data in such a way that it is suited for the task at hand. Once the neural network has learned such a representation of the data in a supervised or semi-supervised manner, it makes it possible to utilize this representation in the various available tasks for renewable energy. In our chapter, we present different techniques to obtain appropriate representations for renewable power forecasting tasks, showing the similarities and differences of deep learning-based techniques to traditional algorithms such as (kernel) PCA. We support the theoretical foundations with evaluations of these techniques found on publicly available datasets for renewable energy, such as the GEFCOM 2014 data, Europe Wind Farm data, and German Solar Farm data. Finally, we give a recommendation that assists the reader in building and selecting representation learning algorithms for domains other than renewable energy.

---

J. Henze · J. Schreiber (✉) · B. Sick

Intelligent Embedded Systems Lab, University of Kassel, Kassel, Germany

e-mail: [j.schreiber@uni-kassel.de](mailto:j.schreiber@uni-kassel.de)

J. Henze

e-mail: [jhenze@uni-kassel.de](mailto:jhenze@uni-kassel.de)

B. Sick

e-mail: [bsick@uni-kassel.de](mailto:bsick@uni-kassel.de)

**Keywords** Representation learning · Deep learning · Renewable energy · Timeseries

## 1 Introduction

Forecasting information about future power production or consumption is attracting more and more interest during the last few years, as the energy system in many countries starts transforming from a traditional centralized system to a more decentralized system. This change is taking place because we are introducing a lot of variable energy producers and consumers in the electrical grid. To maintain a stable electrical grid, we need reliable information about future power production and consumption. This information allows the planning of energy storage and consumption accordingly, so that generation and consumption are balanced.

In day-ahead<sup>1</sup> power forecasts, especially in renewable power forecasting, we often rely on NWP models and use information, e.g., about future wind speed or temperature, to forecast future production and consumption of power. Currently, there are predominantly two separate approaches: The first one is to create a physical model based on the characteristics of, e.g., a wind farm or a household. The second approach is to use machine learning (ML) models, such as artificial neural networks (ANNs) or support vector machines, to perform the task of forecasting future consumption or production based on NWP data.

There is an ongoing debate on whether and when it is better to implement either physical models or ML algorithms. The advantage of the physical models is that they are strongly modeled for a particular type of household or specific renewable power plant, such as a wind turbine, and are applicable without utilizing historical data. The advantage of ML models is that we do not need to know the particular physical characteristics of the renewable power plant, wind turbine, or the building we want to model. The ML model learns a function that maps the input data to our desired output, i.e., NWP data to the power time series. Of course, this data-driven approach, makes models adapt to (noisy) input data, but reduces or even disables the interpretability of such models. However, ML models, especially deep learning (DL) models, are capable of producing smaller errors when forecasting power time series than their physical counterparts [6].

Even though ML algorithms profit from learning a mapping between the input and the target data, they still greatly benefit from *feature selection* and manually *engineering* essential features from the set of all inputs beforehand [4]. Typically, humans or so-called filter and wrapper algorithms need to select the best input features for the task. Finding the most important features can be quite slow and often involves a lot of background knowledge about the solution of the task one wants to solve. Similar challenges arise when *feature engineering* approaches are used to create additional features for the ML tasks by using domain knowledge [4, 30]. The

---

<sup>1</sup>Power or load forecasts of the upcoming day.

whole task of selecting and engineering features is tedious and can take up several selection, evaluation, and engineering cycles. Therefore, the overall process is quite labor intensive, and requires a great deal of computational capacity.

*Representation learning (RL)* tries to overcome these disadvantages as a specialized field of ML that exploits an automatic data-driven feature engineering and feature selection approach. During *RL* we learn *latent (or hidden) features*. Latent features describe the data with sufficient accuracy and often provide the distribution underlying the original input features, which helps the ML task to perform better. Finding and constructing the latent features is also referred to as *feature extraction* [28]. *RL* can be seen as an advancement of manual feature engineering and selection, as we do not need to employ domain knowledge to select essential features, or to come up with mathematical models describing relations between features. Instead, during *RL* we may employ a deep ANN that learns latent features about our input data.

A latent feature model obtained with the help of *RL* often makes it possible to further improve those tasks, that use the latent feature representation as input. These tasks (using the latent features) do not necessarily need to be forecasting a power time series but can also be the classification of images, or prediction of a car's trajectory using different sensory inputs. To present concepts from the field of *RL* which are applicable in the field of power time series forecasting and other domains, this chapter aims to:

- Discuss major challenges in power time series forecasts,
- present *RL* for power time series,
- show the influence of *RL* on power time series forecasting by introducing evaluation methods for *RL*, and
- provide various examples.

As this chapter will introduce the general concepts of *RL* for time series, we do not include a comparative study with the traditional techniques of feature selection and feature engineering.

The remainder of this chapter is structured as follows: Sect. 2 defines a forecasting task based on three examples of power time series. Section 3 explains feature extraction in more detail, introducing traditional algorithms as well as deep architectures for *RL*. Section 4 proposes several evaluation strategies for *RL* in renewable power forecasts based on the previous mentioned definitions. Section 5 shows several examples for *RL* in power time series forecasting. It utilizes the algorithm and evaluation measures to provide examples of *RL* in power time series forecasting. Section 6 concludes this chapter and gives some advice on how to apply *RL* to other ML problems. It also provides insights on how to design your *RL* network and select appropriate parameters.

## 2 Regression in Power Time Series Forecasting

This section defines time series forecasts in the context of renewable energies. Challenges for those time series are introduced based on those definitions.

## 2.1 Renewable Power Time Series Forecasting

We use the term *power time series* for three different types of data here: wind, solar, and load time series. The whole process to forecast the targets of the data is typically a two-step approach.

The first step involves forecasting the weather features [5] with a typical time step of up to  $k = 72$  h in the future. Some important input features provided by the NWP for power time series forecasting are: Wind speed, air pressure, wind direction, temperature, humidity, solar irradiance, rainfall, snow coverage, and much more that can be selected. Creating these features based on NWP models is computationally expensive and that is assumed to be a given in this chapter. The second step maps the set of weather features to the generated power of a wind power plant, the load of a household, or the output generated by a solar power plant. Forecasting a power time series in the second step, e.g., using neural networks, a Support Vector Machine, or a linear regression, involves finding a regression model mapping the set of input features to the power.

While the second step can be considered as a “classic” regression problem without modelling a specific time dependency, the overall process including the first step is referred to as a time series problem [5]. However, in most cases the second step is also considered as a time series problem, as it includes so-called time-shifted features (explained later in this section) that improve the forecast quality.

Correspondingly, both kinds of data, the NWP and the generated power, are time series consisting of an ordered list of tuples:

$$T = \{(t_0, \mathbf{x}_0), \dots, (t_k, \mathbf{x}_k), \dots, (t_n, \mathbf{x}_n)\}, \text{ with} \\ \mathbf{x}_k = (x_0, \dots, x_D). \quad (1)$$

Each of the tuples consists of a timestamp  $t \in \mathbb{R}$  and a feature vector  $\mathbf{x} \in \mathbb{R}^D$ , which gathers all  $D$  data points for that time step. In the case of the power or load time series,  $\mathbf{x}$  is a scalar, i.e.,  $D = 1$ . In case of an NWP time series, it is an pertinent mixture of the items mentioned in the list of weather features. Data samples can be equidistant in time or not. For simplicity and a more straightforward explanation, we assume the former here.

In power and load time series forecasting often something called time-shifted weather features are introduced. Such time-shifted features are weather features of a previous or future time step. Taking future values into account is possible as features from the NWP are themselves forecasted. Consider predicting power at time step  $t_k$ , by additionally including weather features  $(t_{k-1}, \mathbf{x}_{k-1})$  and  $(t_{k+1}, \mathbf{x}_{k+1})$  the forecast error is reduced by introducing the time dependency for previous and future time steps.

As outlined above, during regression at step two, we try to find a function that maps the data of one time series to the data of a second one. In our case, we try to find a function that maps the NWP time series to a power time series. As seen in

Eq. (2), in the case of a linear regression we want to find a set of weights  $\mathbf{w}$  that, multiplied with the input NWP data  $\mathbf{x}$ , results in the current power, allowing for a particular variance of  $\varepsilon$  [24, p. 19]:

$$\text{Power}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + \varepsilon. \quad (2)$$

When doing regression with (deep) neural networks, we try to learn the parameter of a neural network to be able to map our inputs to the appropriate output. Neural networks blow up the linear regression task of Eq. (2) to several of these regression models that are hierarchically combined (including non-linear activation function) with different inputs from different layers. With the help of the neural network, we are also capable of learning even non-linear relations between the NWP data and the power time series, as we can allow for non-linear activation functions, such as the logistic function.

## 2.2 Challenges of Power Time Series Forecasting

The significant challenges in power time series forecasting arise due to the aforementioned transformation of the energy system. The former centralized power grid is changing to a more decentralized grid [22]. Those decentralized grids often have renewable power plants close to where energy is consumed. This introduces significant challenges for the power grids as a whole but also generates new challenges regarding the forecasting.

As we have more and more renewable power plants connected to the grid, we cannot rely on forecasting an aggregated power output, e.g., for all wind power plants in a region, but we must forecast for each power plant individually. Depending on their connection to the power grid, each plant influences a specific part of the local grid [18]. These challenges get even more complicated with home mounted solar panels, or electric vehicles charged at home.

Another challenge arises due to smart grids and their smart measuring infrastructure [13]. This kind of measuring infrastructure increases the amount of available data that needs to be processed by our models, whether they are machine learning or traditional models. In addition to data complexity, additional (smart) measuring infrastructure allows us to create more detailed models of our power grid.

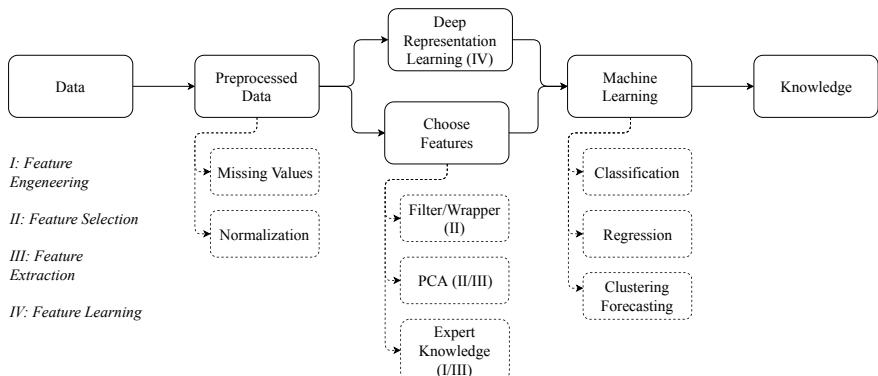
With an increased amount of available data and an increased amount of power plants that need to be forecasted, we have to think about algorithms that allow for easy integration and processing of this vast amount of data. Additionally, the algorithms need to be able to adapt the embedded knowledge to different power plants. One solution to these challenges can be representation learning and based on top of this even multi-task learning.

### 3 Foundations of Representation Learning

Creating a representation or selecting relevant features such as sensor data from complex data can improve the performance of an ML algorithm [2]. The extraction and selection of features help the ML algorithm achieve better results for a particular task when compared to using the raw data directly [3, p. 268]. The research area concerned with determining suitable features is called *representation learning* [2, 14]. Typically, the representation is either in a higher dimensional or a lower dimensional space compared to the original input. In this chapter, we are especially interested in the latter case for deep learning-based methods, as such a representation reduces the computational effort once the representation is learned. Also, for some deep architectures, a lower dimensional representation is more suitable for the training, as discussed in detail later. However, some presented traditional algorithms, such as *kernel* principal component analysis (PCA), first transform the data in a higher dimensional space and then reduce it to a lower dimensional space in a second step.

In the context of ML, the dimensionality of feature reduction is the process of extracting or selecting the important features  $k$ , out of  $n$  input features, where typically  $k$  is smaller than  $n$ . Further, this reduction process aims to find a set of  $k$  features that are non-redundant and informative. Typically, dimensionality reduction techniques are either from the field of feature extraction or the field of feature selection, yielding the following advantages:

- Reducing the computational effort for training the ML model,
- improving the forecast quality, and
- limiting the number of features to informative ones that ideally allow for an interpretation of the model’s behavior.



**Fig. 1** A basic overview of the data mining process with examples of different methods. The methods are annotated with the terms features engineering, selection, extraction, or learning. Note that feature reduction can be part of I-IV

Figure 1 shows a general data mining process annotated with the terms feature engineering, selection, extraction, and learning. Some items have one or two terms, as some of the algorithms do both feature selection and feature extraction. We describe our view on the different terms of feature engineering, selection and extraction, and how we apply them throughout this chapter.

In feature selection, we reduce the number of given features using approaches such as filters and wrappers. Both methods select a subset of  $k$  features from the set of all features  $n$  [20]. Filters are, e.g., information theory-based methods that allow for selecting relevant features based on entropy. Filters are independent of the ML algorithm, while wrapper algorithms are dependent on the ML algorithm and select features based on the best evaluation of the algorithm and the current set of features.

Even though filters and wrappers allow the reduction of the number of relevant features, feature engineering is still a crucial concept to obtain good forecast quality [4]. In feature engineering, we create features in a way that they help the ML algorithm to improve its performance. This engineering is done either by a human expert with domain knowledge or automatically by an algorithm. In the latter case, it is called representation learning or feature extraction.

*RL*, also called feature learning, is a field of methods to derive features that are most relevant to an algorithm. Often this process is described as determining latent or hidden features that explain the process that underlies the data. By learning these latent features, ideally, superior forecast performance over manual feature engineering and filter and wrapper-based methods is achieved. Further, by reducing the number of features through *RL* we reduce computational effort at the same time.

In areas such as vision and natural language processing deep learning-based representation learning methods improve the forecast quality dramatically compared to traditional approaches and manual feature engineering based on domain knowledge. Between 2012 and 2015, the classification accuracy on the *ImageNet-2012* dataset improved from an error rate of 16.4–3.57% by utilizing deep learning-based representation learning methods [1]. Section 3.2 explains some of those state-of-the-art deep architectures for various domains.

To compare those deep architectures to traditional dimensionality reduction, we give a brief overview of PCA, kernel PCA, and explain the concepts of the wrapper and filter approaches in Sect. 3.1 in detail.

### 3.1 Traditional Dimensionality Reduction Techniques

In the following, we highlight some traditional feature reduction techniques. They are traditional in the sense that they have been well known for a long time, but they are also limited in their potential solutions for the extraction and selection of (latent) features compared to deep learning-based methods.

For example, filter and wrapper methods (Sect. 3.1) can only limit the number of relevant features but are not capable of extracting (relevant) features that explain

the underlying distribution. Respectively, filter and wrapper methods are feature selection methods, but not feature extraction techniques.

In contrast, PCA, (Sect. 3.1) allows latent features to be extracted. These features are derived based on the assumption that important features are the directions of the largest variance in the original feature space. For additional details on feature reduction methods, we refer to [2, 20, 32].

**Filter and Wrapper:** Filters are approaches that select a set of relevant features based on a given measure, whereas the size of the set can vary. Typically, they use different evaluation measures. Typically, we differentiate between similarity, information theoretical, and statistical measures [20]. After selecting a suitable measure, the features are evaluated and ranked according to the selected measure. Afterward, the algorithm or the human selects the most relevant features. This process provides an interpretable selection of features with small computational effort in comparison to other dimensionality reduction techniques. Therefore, it scales well with the number of features. The disadvantages are:

- Filters often select redundant features,
- filters ignore the relation to the ML algorithm, and
- filters only reduce the number of *visible* features and are not determining latent features.

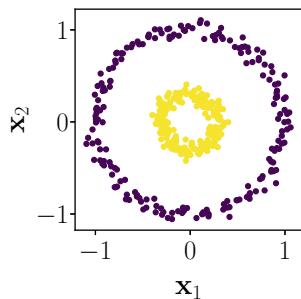
While filters operate independently from the ML algorithm, wrapper algorithms depend on it [10]. In particular, they select the features by iteratively training the ML algorithm on  $i$  subsets of the  $n$  features, then evaluate the performance of the ML model and select the feature set that performs best. In sequential feature forward selection, one starts with an empty set of features. The feature that improves the ML algorithm the most is added to the set of relevant features iteratively, until a pre-defined number of  $k$  features is selected [9]. Other methods use an iterative approach starting with the set of all features and then successively removing the least important feature. As a result, the effort rises quickly with the number of features and extracting latent features is not possible.

**Principal Component Analysis:** PCA is an algorithm that is designed to extract orthogonal features that are linearly uncorrelated. Therefore, PCA assumes that important features have a high variance. The individual steps of the algorithm are:

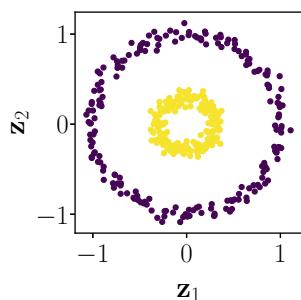
1. Remove the mean from all features.
2. Calculate the covariance matrix of all original features.
3. Calculate the eigenvalues and eigenvectors of the covariance matrix.
4. Sort eigenvectors by their eigenvalues (highest eigenvalue corresponds to the highest variance in the direction of the corresponding eigenvector).
5. Select  $k$  highest eigenvalues for dimensionality reduction.
6. Transform the original input data into a new features space using the eigenvectors.

The transformed features are also called *principal components*, and they are the hidden features extracted by the algorithm.

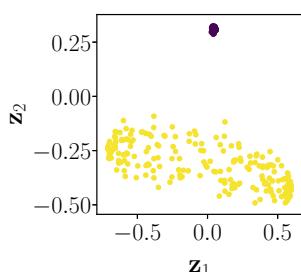
**Fig. 2** Input of kernel PCA with two colour coded class labels



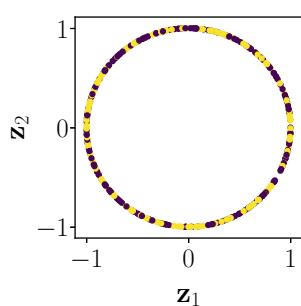
**Fig. 3** Output of kernel PCA, with a linear kernel



**Fig. 4** Output of kernel PCA, with an RBF kernel



**Fig. 5** Output of kernel PCA, with a Cosine kernel



However, often it is beneficial to apply a non-linear PCA, e.g., a kernel PCA. Non-linearity is beneficial if the input features do not follow a linear pattern, as in Fig. 2. In this case, the linear PCA is not capable of finding essential components, as shown in Fig. 3. Therefore, kernel PCA implicitly calculates the covariance matrix of a higher dimensional representation of the input. The well-known *Vapnik–Chervonenkis theory* [35] states the effect that data transformed into a higher dimensional span often provides linearly separable features.

Therefore, we first transform the input data with the *kernel function* into a higher dimensional representation. In the second step, the *kernel* calculates the dot product of the transformed data to obtain the covariance matrix [3, 29, 36]. This combination of non-linear transformation and dot product is referred to as the *kernel-trick*. Once the covariance of the transformed data is calculated, the PCA algorithm is applied. We can interpret the resulting eigenvectors of the kernel PCA as projections from the higher dimensions onto the principal components. After applying the kernel PCA we often obtain better features, as seen in Fig. 4.

Each kernel has a different characteristic, e.g., the higher dimensional features obtained the radial-basis function (RBF) kernels yielding infinite dimensionalities [3, p. 297], while other kernels have different characteristics. Thus, it is important to utilize the kernel that is most suitable for the data. Figure 2 shows the input to the different examples of PCAs. The input has two features,  $x_1$  and  $x_2$ . The circular data presented here has two color-coded labels to indicate a reasonable and non-reasonable transformation concerning the class label.

Figure 3 shows a non-reasonable result, where the linear PCA is not capable of extracting meaningful latent features  $z_1$  and  $z_2$ . In Fig. 4 we see the results of a RBF kernel applied to the input data. We observe that a single PCA component ( $z_2$ ) is sufficient to separate the different classes, while in Fig. 5 the separability of the two color-coded classes is even decreased when compared to the original input. For additional information on kernels and in particular kernel PCA refer to [3].

### 3.2 Deep Architectures for Latent Feature Extraction

In this section, we explain deep architectures that allow for latent feature learning. While in the sections above, *traditional* algorithms from the field of feature selection and extraction are explained, this section focuses on modern architectures that permit the extraction of useful features.

We focus on autoencoders that are capable of learning a representation  $\mathbf{z}$  of the original input  $\mathbf{x}$  by constraining the learning process or the representation  $\mathbf{z}$ . In particular, we are interested in methods that allow determining latent features, while reducing the number of features for further processing and keeping the relevant information at the same time [2, 8]. As stated earlier, reducing the number of features to reduce computational effort is an essential concept in feature extraction. Correspondingly, this section focuses on *undercomplete* autoencoders to learn a compressed representation of the data.

*Undercomplete* autoencoders learn a representation  $\mathbf{z}$  which is smaller than the original input  $\mathbf{x}$ . This architecture is contrary to *overcomplete* autoencoders, in which the representation of the data is higher dimensional than the number of input features. Also, undercomplete autoencoders are more common, because once we learned the compressed representation, the computational effort for further processing is reduced in comparison to the original input and the overcomplete representation.

In the following sections, we distinguish between *generative* and *discriminative* models. While generative autoencoders learn the distribution that is most likely to explain the data, discriminative autoencoders learn an efficient data encoding. Discriminative autoencoders are more convenient to implement and sufficient for most tasks. Generative autoencoders have the advantage that we can use them for denoising, imputation of missing values, and sampling data.

Another way to learn latent features are deep belief networks (DBNs), and their underlying restricted Boltzmann machines (RBM). They learn a distribution of the underlying latent features from which new samples of the input data can be drawn. In an *RL* setting, these are pre-trained using contrastive divergence, and afterward, they need to be fine-tuned to the regression or classification task using e.g., stochastic gradient decent (SGD). During this fine-tuning phase, DBN and RBM behave like a normal multi-layer perceptron (MLP) and therefore are not much different than, e.g., a denoising autoencoder (DAE). Due to this similarity we decided not to explain RBM and DBN in more detail.

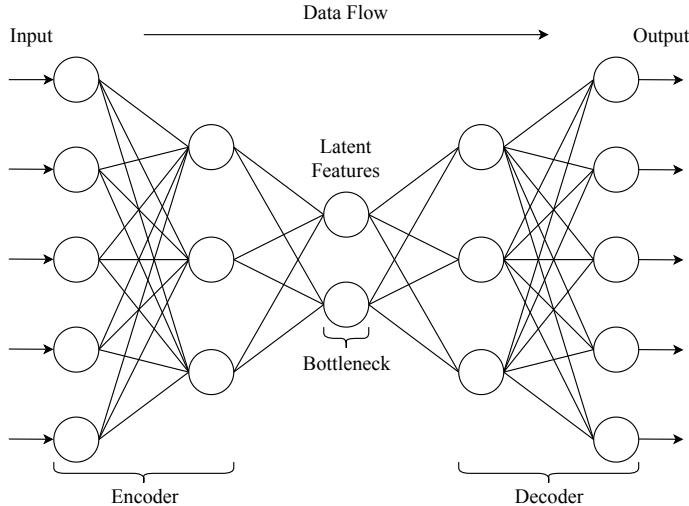
Therefore, in the following sections, we explain three types of discriminative autoencoders giving an introduction to the concept of autoencoders. In the final section, we extend this idea with a generative approach.

**Autoencoder:** An autoencoder (AE) is a variant of an MLP that learns an encoding  $\mathbf{z} = f(\mathbf{x})$  and a decoding  $\mathbf{x} = h(\mathbf{z})$ , where  $\mathbf{x}$  are the input features and  $\mathbf{z}$  is the encoded version of  $\mathbf{x}$ . Due to the reconstruction of  $\mathbf{x}$  from  $\mathbf{z}$  the input and output layers have the same size [8].

Even though AE architectures are diverse, probably the most common architecture is the undercomplete AE as shown in Fig. 6. Undercomplete autoencoders reduce the dimension in each layer starting from the input layer. This side is called the encoding side, learning the mapping to encode  $\mathbf{x}$  with a function  $f$ . At the center, also called the *bottleneck* of the AE, the layers are mirrored to produce the decoding side of the AE to reconstruct  $\mathbf{x}$  with a function  $h$ .

That is, the AE is trained to reconstruct the input on the output side. The idea is that the bottleneck serves as a feature extractor of the input data. Due to the reduced dimensionality at the bottleneck, the AE is forced to learn latent features or an efficient encoding that is sufficient to reconstruct the original features. In particular, the smaller dimension of  $\mathbf{z}$  compared to  $\mathbf{x}$  assures that the MLP is not learning the identity function. The following function typically describes the objective of the training [8]:

$$L(\mathbf{x}, h(f(\mathbf{x}))),$$



**Fig. 6** An example undercomplete AE topology. The AE reduces the dimensionality in each layer of the encoder. The latent features representation, at the bottleneck, are the extracted hidden features that are sufficient to reconstruct the original input successively in each layer of the decoder

where  $L$  is a loss function, e.g., a squared loss, that penalizes the dissimilarity between  $\mathbf{x}$  and the reconstruction  $h(f(\mathbf{x}))$ .

After training the AE, we cut the network behind the bottleneck, and attach a conventional ML algorithm. Using the learned encoding as an input to the regression or classification model is similar to using components of a kernel PCA.

It can even be shown that when the decoder is linear, and we use a squared error loss function, the latent features of the AE are in a similar sub-space to PCA. Moreover, by using singular value decomposition, it is possible to reconstruct the original PCA components [27]. The results from [27], details the similarities of PCA and AEs. To be capable of comparing forecast results obtained from AEs with the more advanced techniques such as the nonlinear PCA, we extend the idea of AE to more complex structures utilizing the potential of deep architectures for representation learning even further.

**Denoising Autoencoder:** An undercomplete AE, as described above, learns latent features by going through a bottleneck. We achieve a similar form of restriction by adding a noise term to the input features. DAEs are what is known as regularized autoencoders, allowing similar results, even with overcomplete architectures. In practice, however, an undercomplete DAE is used to minimize the following loss function

$$L(\mathbf{x}, h(f(\tilde{\mathbf{x}}))),$$

where  $\tilde{\mathbf{x}}$  is a corrupted version of  $\mathbf{x}$ . Therefore, we typically draw from a unit Gaussian distribution so that we obtain a matrix with a shape similar to the original input data. In the next step, we scale the random data by a constant  $c$  and add the scaled random data to the original input. A typical value of  $c$  is, e.g., 0.1. By reconstructing the original input features from the corrupted features, the DAE learns the structure of the input distribution. The combination of regularization and undercomplete architecture yields the following properties:

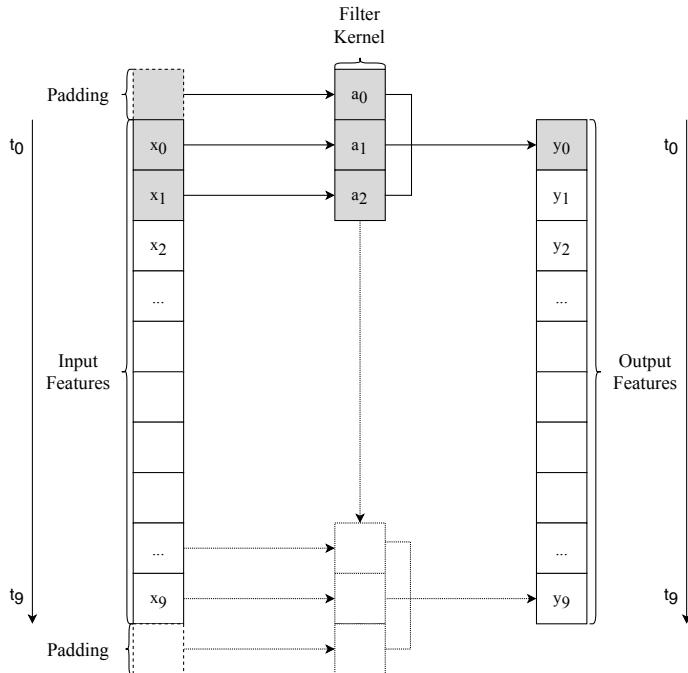
- The bottleneck minimizes the computational effort, once the DAE is trained.
- The regularization removes noise which is common, e.g., in sensor data.
- Bengio et al. [2] reports that DAE often improves the quality of the learned hidden features, by being less noisy.

**Convolutional Autoencoder:** In the following section, we accentuate the specifics of convolutional neural networks (CNNs) in the context of autoencoders [23]. Therefore, we initially review the basic principle of CNNs and explain how it can be used to extract relevant features in the context of time series and continue with an explanation on how to utilize them with AE. Even though two and three-dimensional CNNs are more common in the context of natural language processing (NLP) or vision, we limit our explanation to the one-dimensional CNN. This limitation makes the explanation easier. Further, two and three-dimensional CNNs reuse the filter weights along their higher dimensions, therefore using the information of different features in each convolutional step. This behavior is desired in image processing but might lead to diluted features in time series, such as sensor data or NWP data. These conditions limit the validity of learned features and are not desirable in time series problems.

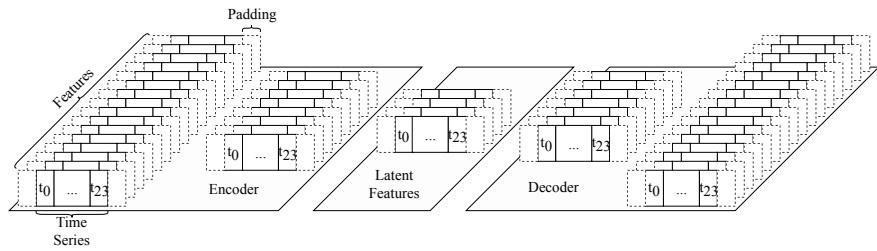
In Fig. 7, we see an example of a CNN layer with a filter size of  $1 \times 3$ , applied with the inner product to the input matrix size of  $1 \times 10$ . We can consider the input matrix as a single input feature with ten time steps. Applying the filter to the input feature matrix mainly extracts features over 10 time steps. By applying the filter size of  $1 \times 3$  with a stride size of 1, we obtain an output shape of  $1 \times 8$ . By including, padding of 1, zeros are added to the beginning and end of the time series, increasing the size to  $1 \times 12$ . When we apply the filter to the time series, we avoid a dimensionality reduction compared to the original input data, and after the filter application, the original input shape of  $1 \times 10$  is maintained.

In case the CNN layer has multiple filters, we obtain an output of  $t_0, \dots, t_9$  for each of those filters. By applying these filters, the CNN is capable of extracting temporal features. These filters are often beneficial as they maintain relations between and future time steps [34, 37].

If this procedure is now repeated in the encoder with the same kernel size, and in each layer, the dimension of a filter's output is reduced, as seen in Fig. 8. Further, by decreasing the number of the filters in each successive layer, the latent feature representation at the bottleneck is obtained. In this bottleneck, the learned feature representation takes simple temporal features into account to represent the data. It is worth noting that we are not using a pooling layer and therefore only decrease the number of input features and convolving over time. A more detailed overview of



**Fig. 7** Example of one-dimensional CNN with a filter size,  $1 \times 3$ , applied to an input time series size of  $1 \times 10$  with additional padding. The additional padding allows to keep the dimension of the time series and to extract relevant information



**Fig. 8** An example convolutional AE topology. The convolutional AE reduces the dimensionality in each layer of the encoder and keeps the temporal information through padding. The latent features representation, at the bottleneck, are the extracted hidden features including a temporal representation of the input to reconstruct the original time series in the decoder

different data and their corresponding  $n$ -dimensional convolutions is given in [8]. On the decoder site, so-called deconvolutional layers are used to reconstruct the original time series. Deconvolutions use an inverse convolution to reconstruct the original time series, e.g., with the same loss function as explained for the vanilla autoencoder in Sect. 3.2. The properties of the convolutional autoencoder can be summarized as follows:

- In time series application one-dimensional filters are sufficient because higher dimensions impose conditions on the order of the features [8, p. 349].
- The combination of CNN and AE forces the algorithm to learn simple temporal features that are sufficient to reconstruct the original time series.

**Variational Autoencoder:** The drawbacks of the discriminative architectures in the previous sections are that they cannot be used to reconstruct missing values or generate new samples. Variational autoencoder (VAE) are a generative approach that extends the idea of a simple autoencoder by adding a constraint on the encoding site to generative properties.

The encoding side is forced to learn the mean  $\mu$  and standard deviation  $\sigma$  of a Gaussian distribution, as shown Fig. 9.  $\mu$  and  $\sigma$  are used to create latent features  $\mathbf{z}$  by sampling from a unit Gaussian scaled with the learned  $\mu$  and  $\sigma$ ; also called reparameterization trick [16]. The scaled samples are used to reconstruct the original features  $\mathbf{x}$  with a function  $h$ . More formally this can be done using a loss function:

$$L(\mathbf{x}, h(q(\mathbf{z}|\mathbf{x}))) - D_{KL}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})),$$

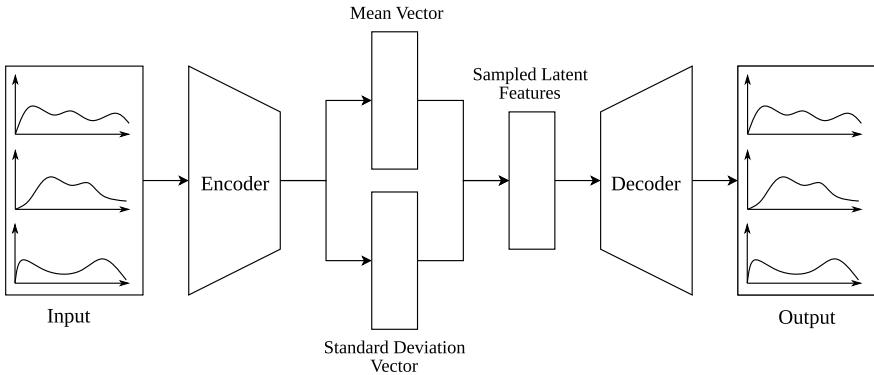
where  $q(\mathbf{z}|\mathbf{x})$  is the scaled version of the unit Gaussian given the current input  $\mathbf{x}$  and the *Kullback-Leibler Divergence*  $D_{KL}$ , see Sect. 4.2, penalizes the deviation between the learned distribution  $q$  from a unit Gaussian.

By applying the reparameterization trick, it is possible to extend the original idea of an AE and achieve the following properties:

- Often the combination of a generative network with an encoder forces the VAE to learn a representation in a much lower dimensional space, see [8, p. 699] and [16].
- The decoder and the latent vector provide a generative framework.

## 4 Evaluation of Representation Learning in Regression Tasks

To evaluate representation learning in power series forecasting, we have to consider three aspects. Firstly, we need to evaluate the overall performance of the feature learner. Secondly, we need to evaluate how the actual regression model performs. As methods for assessing the performance of the feature learner and the regression model are similar, both are detailed in the same section. Thirdly, we need to measure how well our latent features perform. Such a performance measure for latent features



**Fig. 9** An example VAE. The VAE reduces the dimensionality in each layer of the encoder and learns vectors of  $\mu$  and  $\sigma$  from a Gaussian distribution. The vectors are used to scale samples from a Gaussian distribution. The scaled samples are used to reconstruct the original features at the decoding side

needs to reveal how much information our learned representation maintains and how well this representation performs in a forecasting task.

#### 4.1 Evaluation of a Regression Model

To measure the performance of a solution for a regression problem, we can calculate the difference between the model output  $\hat{T}$  and the actual time series  $T$ . Such performance measures are also called time series measures and typically compare the data time series at each point in time. As mentioned in Sect. 2, we work on equidistant time series. Therefore, the root mean squared error (RMSE) or mean absolute error (MAE) are good measures to compare the  $\hat{T}$  and  $T$ . Other measures include the time information of the regarding time series, such as dynamic time warping or time warp edit distance [3, 31].

$$RMSE(T, \hat{T}) = \sqrt{\frac{1}{N} \sum_i^N (\mathbf{x}_i - \hat{\mathbf{x}}_i)^2} \quad (3)$$

$$MAE(T, \hat{T}) = \frac{1}{N} \sum_i^N |\mathbf{x}_i - \hat{\mathbf{x}}_i| \quad (4)$$

The RMSE, as seen in Eq. (3), and the MAE, as seen in Eq. (4), use the data of the model output  $\hat{T}$  and the actual time series  $T$ . They first calculate the difference between both data points. The RMSE then squares this difference, averages the values, and takes the square root of the average. Therefore, RMSE is non-negative

and gives the average distance between the data points and the model output. The MAE similar to the RMSE is non-negative, as it takes the absolute difference of the regressed time series and the original time series, and averages it over all data points. The main difference between those two measures is that the RMSE penalizes substantial differences between the two time series more than then MAE [17].

During the training of the representation learner, we use the RMSE to evaluated the reconstruction loss. Later on, we also employ the RMSE to compare the model quality of different learned representations.

## 4.2 Evaluation of the Learned Feature Representation

Assessing the influence of the features on the regression model output is another way to evaluate the input features of a regression model. Such an assessment uses one set of features, e.g., the learned feature representation, to determine their performance in comparison to another set of features, e.g., the original input features. Furthermore, we can gain information about the amount of compression, contained information, and the features ability to improve the regression. We explain three different measures to compare the feature representations, learned or not. These measures are the compression rate, the Kullback-Leibler Divergence (KLD), and correlation-based measures.

First, we start by giving information about measuring the compression rate we achieve. This information allows us to group algorithms with similar compression rates and to evaluate within these groups.

$$\text{Compression Rate} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}} \quad (5)$$

$$\text{Compression Rate} = \frac{\text{Number of input features}}{\text{Number of latent features}}$$

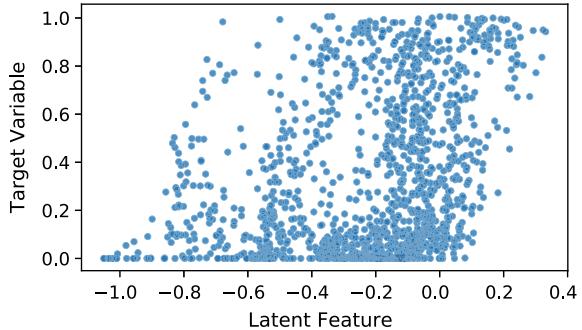
The compression rate as seen in Eq. (5) is the ratio between input data and output data. In our case, we compare the number of features in the input layer to the number of features after the encoding. This comparison allows us to compare several metrics on different datasets grouped by the compression rate.

An essential measure to assess the learned feature representation is the *mutual information* which is based on the KLD. The KLD allows us to measure the similarity of two distributions. The *mutual information* allows us to measure the influence of each latent feature with the regression model output [3]. For simplicity, we limit the explanation to the discrete case of distributions.

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left( \frac{P(x)}{Q(x)} \right) \quad (6)$$

$$MI(X; Y) = D_{\text{KL}}(P(X, Y) \parallel P(X)P(Y)) \quad (7)$$

**Fig. 10** The power production target plotted against one latent feature



The KLD in Eq. (6) is used to compare two discrete distributions  $Q$  and  $P$ . It measures the similarity of the two random variables  $X$  and  $Y$  with  $x \in X$  and  $y \in Y$ . The KLD is not symmetric, i.e.,  $D_{\text{KL}}(P \parallel Q) \neq D_{\text{KL}}(Q \parallel P)$ . If both  $P$  and  $Q$ , are feature representations, we obtain information about their relative entropy. Using the mutual information (MI), see Eq. (7), with  $X$  being the feature representation and  $Y$  the original power time series, we obtain information about how well  $Y$  can be encoded using the current feature representation  $X$  [19], allowing us to compare different learned feature representations regarding their ability to contribute towards the regression model output.

KLD can be used to calculate the MI, or relative entropy of two distributions, e.g., when comparing the distributions in the feature space, or calculating the information loss of a linear model performed with the original input features to a linear model performed with the learned features. This approach is similar to the way t-SNE works [33].

Furthermore, we can apply correlation measures, such as Pearson's correlation coefficient, as shown in Eq. (8). The correlation coefficient quantifies how well our learned features—again, considered as a pair of random variables  $X$  and  $Y$ —linearly correlate with the power time series. Therefore, we identify these feature representations in a linear regression task.

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \quad (8)$$

In addition to the correlation coefficient, we can measure the influence of different latent features using an analysis of variance (ANOVA) [4]. ANOVA allows us to identify the influence of certain features on the power time series. Consequently, creating a measure that identifies if all features of the representation contribute to the regression task or if there are just a few contributing features. With a good feature representation, all features should contribute equally to the regression model [9].

Furthermore, the correlation analysis can also be complemented with simple visualization techniques, such as a scatter plot, as shown in Fig. 10, that allow us to evaluate our learned features against our target variable in the regression task.

## 5 Representation Learning Applied in Power Time Series Forecasting

In this section, we detail the process of *RL* in power time series forecasting. We utilize the information we described in the previous section, to provide a data analysis process for power time series forecasting. The focus of this section is on *RL* and its evaluation for renewable energy tasks. However, we provide generic descriptions so that similar evaluation processes can be defined in other domains.

### 5.1 The Power Time Series

In total, we evaluate three different power time series datasets. All of these datasets are a combination of an NWP model and the measured power production or consumption. These datasets include:

- Europe Wind Farm dataset,
- German Solar Farm dataset, and
- GEFCOM 2014 dataset.

The Europe Wind Farm and German Solar Farm dataset can be downloaded from our website<sup>2</sup> and the GEFCOM2014 is also publicly available online.<sup>3</sup> These datasets make our data quite diverse, and we cover a broad spectrum of power time series forecasting.

**Europe Wind Farm Dataset:** The Europe Wind Farm Dataset consists of the data from 45 wind power plants scattered across Europe. The dataset provides the NWP data as well as the corresponding power output normalized according to the installed capacities. In addition to the available features in the dataset, we augmented the available features using 1h and 2h time-shifted features for wind speed and wind direction allowing time-dependent changes of the future and past weather to be taken into account, see Sect. 2.1.

**German Solar Farm Dataset:** The German Solar Farm dataset consists of the data from 21 photovoltaic facilities in Germany. Their installed nominal power ranges are between 100 and 8500 kW. The PV facilities range from PV panels installed on rooftops to full-fledged solar farms. All these facilities are distributed throughout Germany [6]. Analogous to the Europe Wind Farm dataset, they provide the corresponding NWP and the power time series which are normalized to the corresponding installed capacities. Again, we augmented the available features using 3h time-shifted features for sun position, solar height, clear sky, and radiation.

---

<sup>2</sup><https://www.ies.uni-kassel.de>, last accessed: April 2019.

<sup>3</sup><http://dx.doi.org/10.1016/j.ijforecast.2016.02.001>, last accessed: April 2019.

**Gefcom 2014 Dataset:** GEFCOM, or Global Energy Forecasting Competition, is a dataset based on the GEFCOM2014 forecasting challenge [11]. For the 2014 competition, four different tracks were available, i.e., electric load, electricity price, wind power, and solar power forecasting. In the 2014 challenge, the main task was probabilistic forecasting. In this work, we only use the data of the wind power forecasting track, which is relevant for deterministic forecasts. In the GEFCOM dataset, we did not take time-shifted features into account.

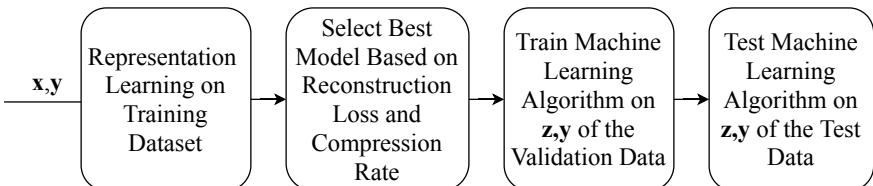
## 5.2 Representation Learning Experiments

In the following section, we explain the *RL* experiments step by step. Figure 11 summarizes the training process of an individual experiment:

- We split the dataset into training, validation, and test datasets.
- We train an *RL* technique with  $k$  hidden features on the training dataset and calculate the compression rate as given in Eq. (5).
- In those cases where we test different variables of hyperparameters, we select the best model based on the evaluation results in the validation dataset. Using the RMSE as evaluation measure allows the reconstruction quality of the original features to be assessed, see Sect. 4.1.
- After selection of the *RL* model, we train the power forecast models on the latent features and power of the validation dataset.
- In the end, the ML models are evaluated on the test dataset.

We repeat the whole procedure for each of the datasets mentioned in Sect. 5.1, our selected *RL* techniques, and between 2 and 9 latent features corresponding to different compression rates, depending on the number of input features contained in the dataset. We focus the evaluation measures on two aspects, as these build the foundation for more advanced techniques such as KLD and correlation-based measures. In the first aspect, we show examples of the relation between reconstruction loss and the compression rate. In the second aspect, we evaluate the forecast error (RMSE) of the models concerning the compression rate.

**Preprocessing of the Data:** We split all of the datasets into training, validation and test sets. This splitting makes it possible to train and select the best model based on



**Fig. 11** Overview of *RL* steps.  $\mathbf{z}$  refers to the latent features and  $\mathbf{y}$  is the power generation

the validation set. The last split, the test, set is then used to do a final evaluation of the regression task on unseen data. For preprocessing, we assure that we normalize the data to have zero mean and a standard deviation of 1. The values of standardization parameters from the training dataset are applied to the validation and test dataset afterward. Further, we normalize the generated power between 0 and 1 by the maximum power generation of each farm. We also avoid categorical input features, because those only roughly describe the weather phenomena but have a considerable influence on the training process of the AE, especially on the reconstruction. Often, when we used categorical features, the AE learned to reconstruct those categorical features and was not capable of reconstructing the nominal weather features from the hidden representation. Therefore, we avoid categorical features in all experiments. By adding the time-shifted features, we allow features with time dependency as those are relevant for time series forecasting. For details refer to Sect. 2.1. In the Europe Wind Farm dataset we use four different time-shifted features.<sup>4</sup> Similarly, four shifted features<sup>5</sup> are added for the German Solar Farm dataset.

**Applied Machine Learning Models:** In the experiments, we always use the same set of hyperparameters for the support vector regression (SVR) and the MLP, the ML models.

For both models, we use the standard parameters given by the *scikit-learn* framework [26]. The SVR uses an RBF kernel and we train it without a hard limit on the iterations. The MLP uses one hidden layer with 100 neurons, *ReLU* activation functions. We train with the Adam optimizer [15] for a maximum of 200 iterations.

**Applied Time Series Measures:** For all of our experiments we use the reconstruction loss and the forecast error, see Sect. 4 for more details. The reconstruction loss allows us to measure the maintained information within our representation. The forecast error allows us to determine if the latent feature representation performs well in a forecasting task. These two measures are the most intuitive and well-known ones in forecasting tasks.

**Guidelines for the Training:** This section examines the training process to provide a guideline for use in other domains.

As previously mentioned, we select the best performing model based on the reconstruction loss of the validation dataset. After selecting the *RL* model we encode the input features  $\mathbf{x}$  to latent features  $\mathbf{z}$ . Afterward,  $\mathbf{z}$  acts as the new input to the ML model. By using the validation dataset, we minimize the risk that the *RL* model is overfitted to the training data and as a result, it does not generalize well on unseen data. Furthermore, using the validation dataset assures, that the ML learns data not seen during the training of the *RL*. We evaluate the final model using the test dataset and the measures introduced in Sect. 4.2.

In our experiments, one significant difference between the evaluated *RL* models concerns the latent features. In the case of PCA, the latent features are the number

---

<sup>4</sup>WindSpeed100m, WindSpeed10m, WindDirectionZonal100m, WindDirectionMeridional100m.

<sup>5</sup>SolarRadiationDirect, SolarRadiationDiffuse, SunPositionSolarHeight, SunPositionSolarAzimuth.

of components which first transform the input data into a higher dimensional space by using kernel functions. In the case of the AEs and DAE, see Sect. 3.2, the number of latent features is equal to the neurons in the bottleneck. In the case of a VAE, see Sect. 3.2, the number of latent features is equal to the number of learned  $\mu$  of the normal distribution, as in practice  $\mu$  is sufficient as expectation. For the convolutional autoencoder (CAE), see Sect. 3.2, since the feature representation includes information about 24 time steps of the NWP. It is important to note that in none of the deep learning-based architectures have we transformed the data into a higher-dimensional feature space. Also note that including the time-shifted features in the input forces the *RL* model to determine latent features that include the time dependency of input features.

In future applications, it might be necessary to select the *RL* model with a specific compression rate, see, e.g., Fig. 12, depending on the reconstruction loss. Selecting an appropriate model with a specific compression rate can reduce the computational effort of certain ML models, as their computational effort increases with the number of input features.

In the following sections, we describe our experimental results and give details on the training procedure and the hyperparameters for *RL* in power time series forecasting. We show how the different proposed *RL* approaches perform compared with traditional approaches. Therefore, we apply four different types of AEs, as well as linear and non-linear PCA on the dataset to learn and extract new features. We use the latent features as input in a regression forecast model. This model maps the latent features from the NWP data to the power time series. In particular, we are interested in showing the advantages and disadvantages of the evaluated *RL* methods.

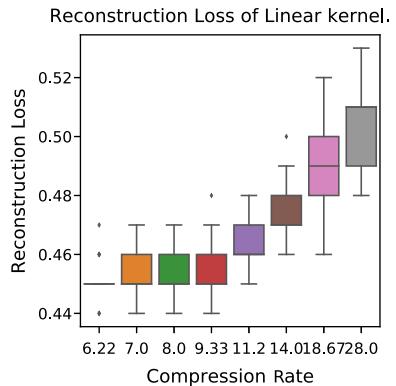
We try to explain everything in a manner that allows for the easy repetition of the experiment in other domains. Therefore, we use the state-of-the-art machine learning framework *scikit-learn* [26] in connection with *pytorch* [25].

In the following section, we first evaluate the traditional feature extraction technique, PCA, in Sect. 5.3. Afterward, we evaluate the *RL* methods for feature extraction in Sect. 5.4. In Sect. 5.6, we discuss the results achieved by both methods. By separating the evaluation for traditional and *RL* methods, we aim to derive recommendations on how to apply *RL* to power time series forecasting.

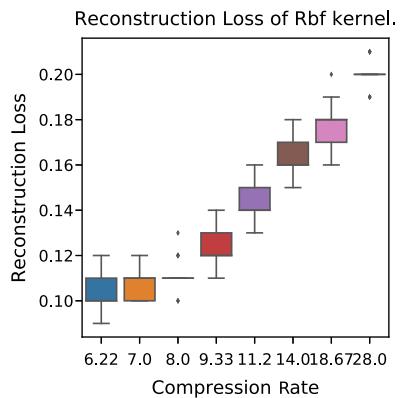
### 5.3 Principal Component Analysis for Feature Extraction

In this section, we highlight the results obtained by PCA. We use PCA as a reference because it extracts hidden features and can reduce their number at the same time, see Sect. 3.1. This extraction and selection process permits comparisons to the deep architecture based *RL* methods. In contrast, filter and wrapper-based methods do not allow for the extraction of new hidden features, see Sect. 3.1. Further, by evaluating different kernels, we show their characteristics concerning the compression rate and the regression task. Assessing these values allows a wide number of representations

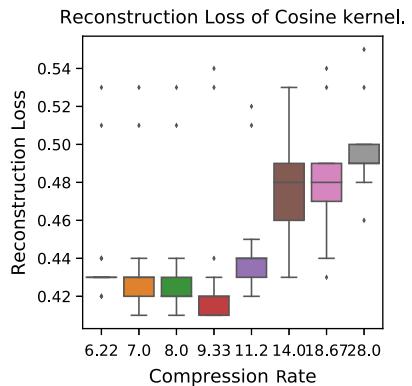
**Fig. 12** Reconstruction loss by linear PCA over different compression rates based on the German Solar dataset



**Fig. 13** Reconstruction loss by *Rbf* PCA over different compression rates based on the German Solar dataset



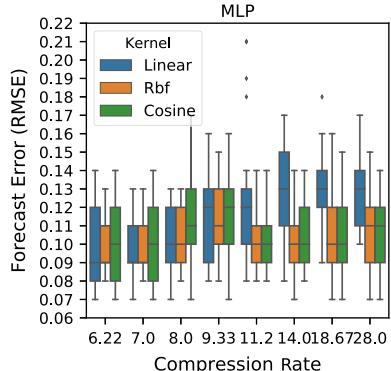
**Fig. 14** Reconstruction loss by *Cosine* PCA over different compression rates based on the German Solar dataset



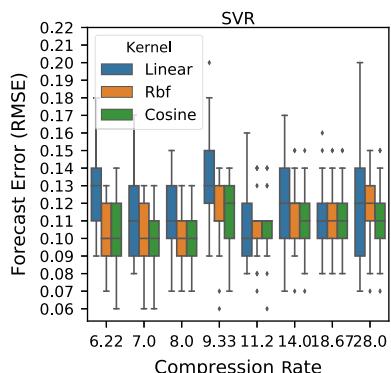
to be compared with the same algorithm, similar to the different deep learning-based representations.

Figures 12, 13, and 14 show the reconstruction loss of three different kernels PCAs applied to the German Solar Farm dataset. In all cases, we observe that when

**Fig. 15** Forecast error of MLP Model applied to latent features obtained from all PCA kernels evaluated for the different compression rates for the German Solar dataset



**Fig. 16** Forecast error of SVR Model applied to latent features obtained from all PCA kernels evaluated for the different compression rates for the German Solar dataset

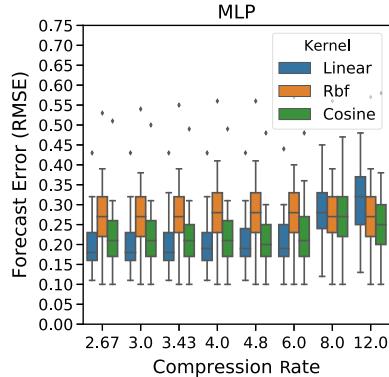


increasing the compression rate the reconstruction loss increases as well. This observation is to be expected, as the decreasing number of hidden features limits the available information when performing a reconstruction. Alternatively, in terms of PCA, the number of components is not sufficient to reconstruct the full variance of the data.

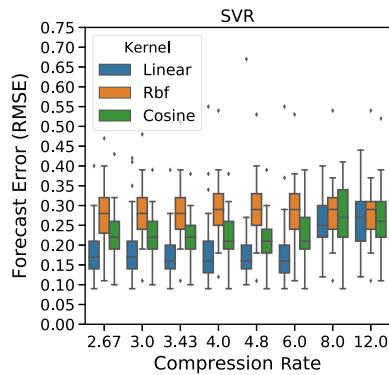
However, the figures presented here illustrate the different behaviors of the applied kernel. In case of the *linear* kernel in Fig. 12 we observe an almost constant reconstruction loss which then increases quickly. In contrast, the reconstruction loss of the *rbf* kernel increases rapidly after a compression rate larger than 8. The reconstruction loss of the *cosine* kernel is roughly constant at a median reconstruction loss between 0.41 or 0.43 until a compression rate of 11.2. The loss then increases up to an RMSE of 0.49. We also note, that for the *cosine* kernel, we have at least two outliers for every compression rate. Comparing all techniques, we observe that the *rbf* kernel has the lowest reconstruction error, followed by the *cosine* and the *linear* kernels.

Figures 15 and 16 summarize the results of the MLP and the SVR. We achieve these results by training the ML model on the extracted features from each kernel PCA for all compression rates. Correspondingly, these figures show the relationship between the forecast error and the compression rate.

**Fig. 17** Forecast error of MLP model applied to all PCA kernels evaluated for the different compression rates for the Europe Wind Farm dataset



**Fig. 18** Forecast error of SVR model applied to all PCA kernels evaluated for the different compression rates for the Europe Wind Farm dataset

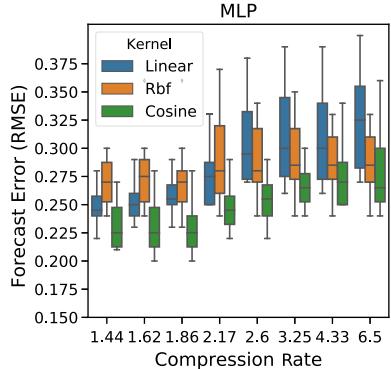


The results show that the linear kernel has the most substantial RMSE deviations. The median RMSE of the linear kernel for the MLP increases with the compression rate. The RBF is the best performing kernel for the MLP, as it has the lowest median RMSE when compared to the other kernels or at least a similar median RMSE. The RMSE for both non-linear kernels behaves similarly to the MLP model, with only slight changes in lower compression rates. The SVR shows a similar RMSE behavior but with more variations throughout the different compression rates.

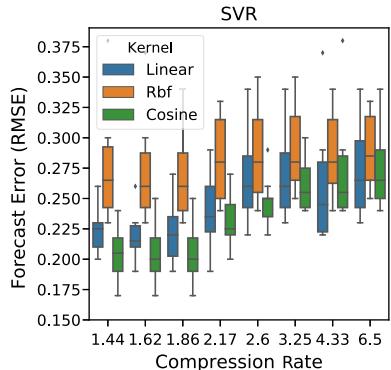
The results for the Europe Wind Farm dataset are shown in Figs. 17 and 18. The compression rates vary between 2.67 and 12.0. The linear kernel has the lowest median forecast error for both ML models on all compression rates up to a compression rate of 8. From a compression rate of 8, the cosine kernel seems to be performing better for all ML models in comparison to the other kernels. It is worth noting that all kernels show some outliers in forecast error.

Figures 19 and 20 show the results of the GEFCOM2014 Wind dataset. Due to the amount of input features, the compression rates vary between 1.44 and 6.5. It can be seen that for most compression rates the cosine kernel performs well for both ML models. The median error of the cosine kernel varies between 0.225 and 0.26 for the MLP model and between 0.21 and 0.26 for the SVR model.

**Fig. 19** Forecast error of MLP model applied to all PCA kernels evaluated for the different compression rates for the Wind GEFCOM2014 dataset



**Fig. 20** Forecast error of SVR model applied to all PCA kernels evaluated for the different compression rates for the Wind GEFCOM2014 dataset

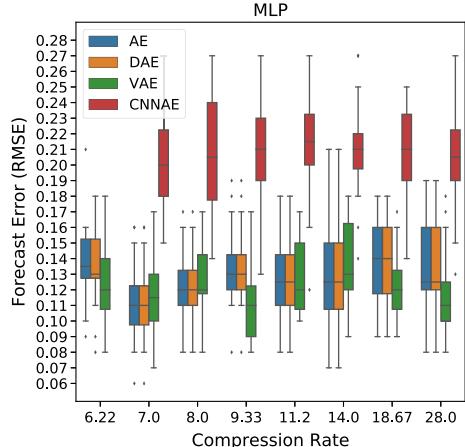


#### 5.4 Deep Architectures for Feature Extraction

Deep architecture based *RL* often provides additional characteristics such as denoising and inferring missing values that are not achievable with other algorithms. However, we need to carefully select relevant hyperparameters and preprocessing steps to obtain good results. This section first gives a comprehensive justification on the hyperparameters followed by the evaluation of the different autoencoder types. Note that the results of the reconstruction loss for the autoencoders are similar to the ones from PCA and are therefore not shown in the results.

**Common Parameters in the AE Experiments:** In many cases, it is complicated to select a set of hyperparameters that permit us to achieve excellent results in deep learning. However, recently, a few techniques (e.g., Adam, Xavier initialization, and batch normalization) have made it possible to achieve excellent results while minimizing the number of hyperparameters that need to be tuned. The following list describes a selection of the important parameters:

**Fig. 21** Forecast error of MLP model applied to latent features obtained from AE, DAE, VAE, and CAE, evaluated for the different compression rates for the German Solar dataset

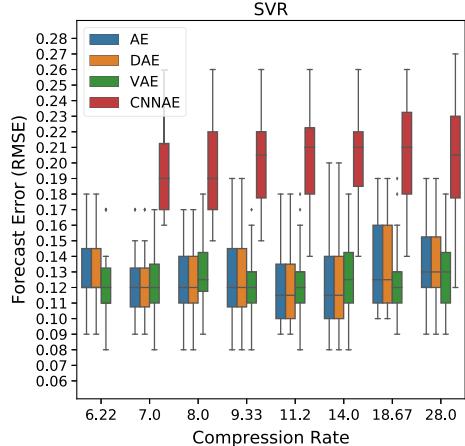


- We vary the number of latent features between 9 and 2. These numbers provide a broad range of compression rates depending on the dataset showing the effects concerning input features.
- For AE, DAE, and VAE, learning rates of 0.001, 0.0005, and 0.0001 are evaluated. For CAE we test the learning rates of 0.01, 0.001, and 0.0001.
- Leaky ReLus are used as activation functions.
- The Adam optimizer is used to train the network.
- In each layer, the number of neurons is reduced by a factor of 0.8 making it possible to create deep nets that successively reduce the number of features to the required number of latent features. Note that another common possibility is first to increase the number of neurons compared to the original input features. In a sense, this would be similar to the transformation of the nonlinear kernel PCA, but we do not consider it.
- Utilizing Xavier as initialization, as a state-of-the-art method to initialize weights, minimizes the risk of exploding gradients [7].
- Similar advantages are achieved by normalizing the input (e.g., avoiding exploding gradients). Therefore, we use batch normalization in each layer [12]. Preliminary examinations showed that using batch normalization for AE, DAE, and VAE achieves at least similar good results as without.

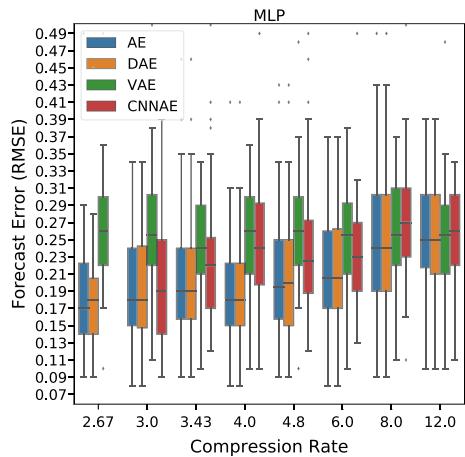
**Summary of the Evaluation Results for the AE, DAE, VAE and CAE Architectures:** The results of the deep architectures for the German Solar Farm dataset are shown in Figs. 21 and 22. In all cases the AE and DAE have a predominantly similar median RMSE and forecast error deviation. Compared to the other *RL* models, the CAE has the highest median RMSE values for both ML models and the VAE has the smallest median RMSE for both ML models, except for a few compression rates. For the MLP, the VAE obtains a median RMSE comparable to the PCA experiment.

Figures 23 and 24 show the results for the Europe Wind Farm dataset. The AE and the DAE have the smallest median forecast error, where the DAE has a slightly smaller

**Fig. 22** Forecast error of SVR model applied to latent features obtained from AE, DAE, VAE, and CAE, evaluated for the different compression rates for the German Solar dataset



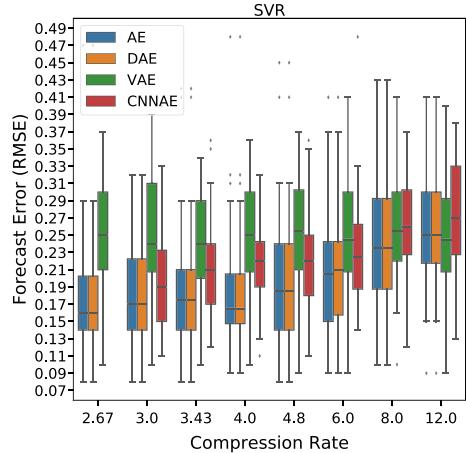
**Fig. 23** Forecast error of MLP model applied to latent features obtained from AE, DAE, VAE, and CAE, evaluated for the different compression rates for the Europe Wind Farm dataset



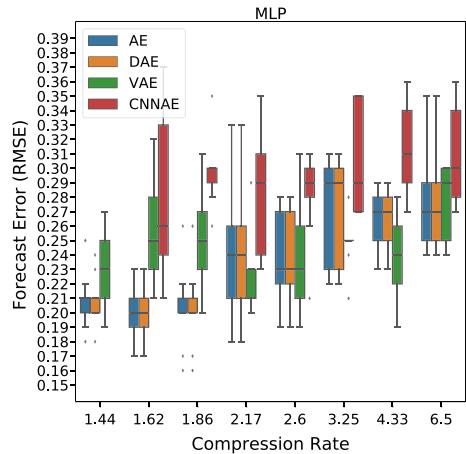
standard deviation. For smaller compression rates the two models have similar results to those of the PCA, but slightly improve for higher compression rates. The VAE has a similar forecast error for all compression rates and forecast models. The CAE performs a bit worse than the AE and DAE. All *RL* models produce some outliers regarding the forecast error.

The results of the GEFCOM2014 Wind dataset are shown in the Figs. 25 and 26. The AE and DAE perform similarly on both forecast models. Both of these *RL* models also have the best performance for smaller compression rates. For the more conspicuous compression rates, the VAE performs the best. The CAE is the worst performing model with an high overall RMSE.

**Fig. 24** Forecast error of SVR model applied to latent features obtained from AE, DAE, VAE, and CAE, evaluated for the different compression rates for the Europe Wind Farm dataset



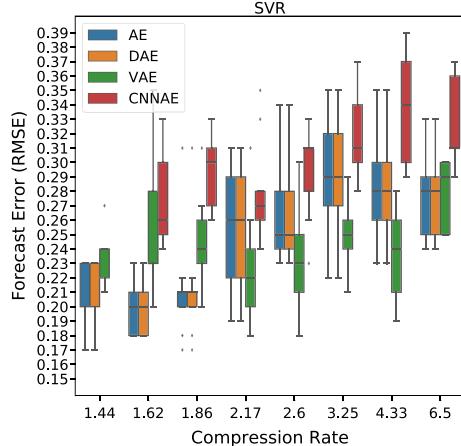
**Fig. 25** Forecast error of MLP model applied to latent features obtained from AE, DAE, VAE, and CAE, evaluated for the different compression rates for the GEFCOM2014 Wind dataset



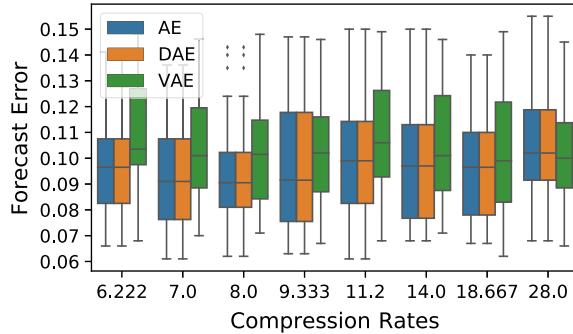
## 5.5 Fine-Tuning for Power Forecasting

In the previous section, we use the learned feature representation directly and train ML models based on those. However, fine-tuning provides a more sophisticated approach towards forecasting power time series with previously learned AEs. The problem with the previously mentioned approach is that the autoencoder's weights are not optimal for the forecasting problem. However, we optimize the weights to reconstruct the input features from a smaller feature representation. Apparently, due to this unsupervised learning process for the autoencoder, the learned representation of the autoencoder might not be ideal for forecasting power time series. Fine-tuning tries to overcome this problem by partly updating the weights of the trained AE for the supervised task of power time series forecasting.

**Fig. 26** Forecast error of MLP model applied to latent features obtained from AE, DAE, VAE, and CAE, evaluated for the different compression rates for the GEFCOM2014 Wind dataset



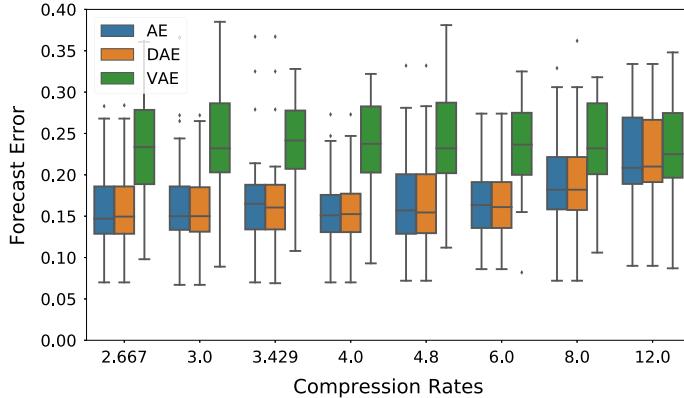
**Fig. 27** Forecast error of fine-tuned MLP model based on AE, DAE, VAE, and CAE. Evaluated for the different compression rates for the German Solar dataset



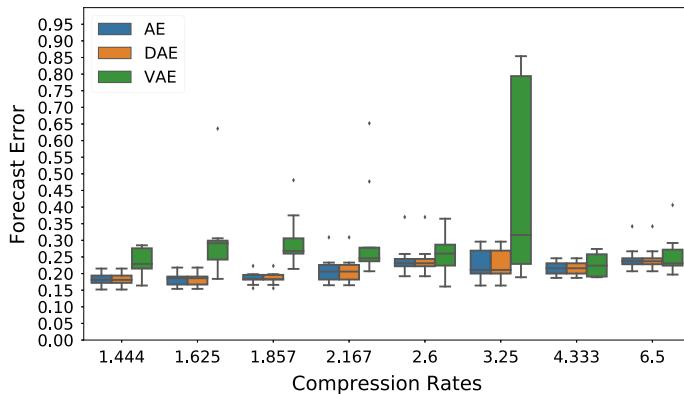
For our scenario, this means that we partially re-train the previously learned encoder. First, we add a linear layer, equal to the MLP from the previous experiment, to the bottlenecks of the AEs. Correspondingly, we have a hidden layer with 100 neurons connected to the number of latent features of the encoder. Furthermore, we add an output layer to map the 100 neurons to the power time series. Second, in the process of fine-tuning, we restrict the adaptation of weights to the last four layers: The output layer of the MLP, the hidden layer of the MLP, and the last two layers of the already trained AE. This restriction minimizes the training effort and makes the best use of the previously learned representation.

Figure 27 illustrates the results of the procedure described above. In contrast to the default parameters of *scikit-learn*, we use a weight decay of 0.2 as described in [21] for AE and DAE. Furthermore, we train for 2000 epochs with a batch size of 2048.

The fine-tuning improves the previous results shown in Fig. 21, even though we train the same number of neurons compared to the previous experiment on the MLP model. By using fine-tuning, the median RMSE improves to values between 0.09



**Fig. 28** Forecast error of fine-tuned MLP model based on AE, DAE, VAE, and CAE. Evaluated for the different compression rates for the Europe Wind dataset



**Fig. 29** Forecast error of fine-tuned MLP model based on AE, DAE, VAE, and CAE. Evaluated for the different compression rates for the GEFCOM2014 Wind dataset

and 0.1 for all compression rates for the encoder of the AE and DAE. The median RMSE of the VAE is about 0.01 higher in comparison to the other fine-tuned models.

Comparing to the PCA results in Fig. 15, we can see that for compression rates higher than 6.2 the finetuned AE and DAE have smaller median RMSE values. Further, in all cases, the fine-tuned models have at least a similar small standard deviation of the forecast error.

The Europe Wind Farm dataset achieves a more substantial improvement, as seen in Fig. 28. The median RMSE decreases to 0.15 up to a compression rate of 6 and then increases to 0.19 for the best models. These results are an improvement over the best PCA, with the smallest median RMSE of 0.1755 for lower compression rates.

We obtain similar improvements with the GEFCOM2014 Wind dataset, as shown in Fig. 29. For almost all compression rates the median RMSE is around 0.175. These

results are an improvement over the best PCA, with the smallest median RMSE of 0.225.

## 5.6 Key Insights

In the previous sections, we explained experiments and well-known algorithms aiming to use representation learning techniques for power time series forecasting, see Fig. 11. For the feature extraction process, we started with the well-known PCA. One significant difference compared to the deep architectures is, that the kernel PCA first transforms the input data into a higher dimensional space. The eigenvectors from the kernel PCA can be considered as a projection from the higher dimensional space, permitting reasonably good forecast results for all evaluated datasets.

This concept is contrary to the architectures and methods used in the first autoencoder experiment, where we directly reduce the dimensionality by a factor of 0.8 in each layer. The final dimension, at the bottleneck, is similar to the number of components used in the kernel PCA experiment. Therefore, we directly compare the forecast error of the autoencoder to the results obtained from the kernel PCA based on the different compression rates.

In the first deep learning experiment, the AEs perform slightly worse or similar to the kernel PCA. These results might be due to the transformation into a higher dimensional space by the kernel PCA. The *Vapnik–Chervonenkis theory* [35] can describe these effects.

However, by utilizing fine-tuning, we are capable of achieving results superior to the kernel PCA by optimizing the already learned representation to its task.

Overall the GEFCOM2014 dataset has the worst results compared to the other datasets. This result might be related to the missing time-shifted features that introduce the relevant time dependency required for time series forecasts.

## 6 Conclusion

This chapter proposed to introduce *RL* in the context of power time series forecasting. We did this by introducing traditional pre-processing methods such as feature selection and feature engineering. Instead of manually finding useful input features for our ML task, we applied *RL* algorithms, especially ANNs with deep architectures, to learn latent features. We additionally showed how to evaluate the representation with and without successive ML algorithms to find good *RL* models. In most cases, we differentiated between distribution-based measurements and measurements applied to the output of ML models trained on the feature representation. In the end, we showed various examples of *RL* in the field of renewable power forecasting.

Representation learning can be seen as the starting point for every ML task, as it obviates the necessity of domain knowledge and permits machine learning models

to increase accuracy. Even though we focused on forecasting power time series in this chapter, we provided generic concepts when possible and conclude this chapter by providing insights on how to build a successful *RL* model.

## 6.1 How to Build a Representation Learning Model

As we have shown in previous sections, *RL* replaces feature engineering and feature selection in the data mining process. Therefore, it precludes the necessity of doing manual feature engineering and selection. In this section, we propose some simple conventions to apply *RL* to regression or forecast applications as conveniently as possible.

First of all, it is most likely that the choice of AE will not matter in regard to performance, as all of the chosen *RL* techniques perform similarly to or even better than, our baseline. To select the best *RL* for your problem the characteristics of your data need to be identified. Depending on those characteristics a corresponding AE can be chosen. Some proposals, as mentioned in Sect. 3, are:

- CAE is good at maintaining and extracting temporal features and should be combined with, e.g., recurrent networks to make the best use of those features.
- VAE can infer missing data.
- DAE can reduce the amount of noise in the latent features.

With the help of this information, and knowing the characteristics of the data, it is possible to identify the type of autoencoder that is needed. The next step is to choose the ML technique to map the latent features to the target data. We have shown that even simple algorithms, such as linear regression, SVR, or MLP will improve the essential metrics for the problem. In the case of MLP models, one should also think about adding a fine-tuning step for a task-specific adaption of the weights, as shown in Sect. 5.5. This additional step permits the ANN to specialize even further towards the goal of the final ML task and improves the initial results considerably.

**Acknowledgements** This work was supported within the project Prophesy (0324104A) and c/sells RegioFlexMarkt Nordhessen (03SIN119) funded by the BMWi (Deutsches Bundesministerium für Wirtschaft und Energie / German Federal Ministry for Economic Affairs and Energy).

## References

1. Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Van Esen, B.C., Awwal, A.A.S., Asari, V.K.: The history began from AlexNet: a comprehensive survey on deep learning approaches. CoRR [arXiv:1803.01164](https://arxiv.org/abs/1803.01164) (2018)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. CoRR [arXiv:1206.5538](https://arxiv.org/abs/1206.5538) (2012)
3. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)

4. Domingos, P.: A few useful things to know about machine learning. *Commun. ACM* **55**(10), 78 (2012)
5. Gensler, A.: Wind power ensemble forecasting. Ph.D. thesis, University of Kassel (2018)
6. Gensler, A., Henze, J., Sick, B., Raabe, N.: Deep learning for solar power forecasting—an approach using AutoEncoder and LSTM neural networks. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics, pp. 002858–002865 (2016)
7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. *AISTATS* **9**, 249–256 (2010)
8. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
9. Freedman, D.A.: Statistical Models: Theory and Practice, 2nd edn. Cambridge University Press (2005)
10. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning The Elements of Statistical Learning, vol. 27. Springer (2017)
11. Hong, T., Pinson, P., Fan, S., Zareipour, H., Troccoli, A., Hyndman, R.J.: Probabilistic energy forecasting: global energy forecasting competition 2014 and beyond. *Int. J. Forecast.* **32**(3), 896–913 (2016)
12. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. *CoRR* [arXiv:1502.03167](https://arxiv.org/abs/1502.03167) (2015)
13. Joy, J., Jasmin, E.A., John, V.: Challenges of smart grid. *Int. J. Adv. Res. Electric. Electron. Instrum. Eng.* **2**(3), 2320–3765 (2013)
14. Khalid, S., Khalil, T., Nasreen, S.: A survey of feature selection and feature extraction techniques in machine learning. In: Science and Information Conference, pp. 372–378. IEEE (2014)
15. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. *CoRR* [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
16. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. *CoRR* [arXiv:1312.6114](https://arxiv.org/abs/1312.6114) (2013)
17. Kirchgässner, G., Wolters, J.: Introduction to Modern Time Series Analysis. Springer, Berlin (2007)
18. Kohler, S., Agricola, A.C., Seidl, H.: dena-Netzstudie II. Technical report, Deutsche Energie-Agentur GmbH (dena), Berlin (2010)
19. Kraskov, A., Stögbauer, H., Grassberger, P.: Estimating mutual information. *Phys. Rev. E Stat. Phys. Plasmas Fluids Related Interdiscip. Topics* **69**(6), 16 (2004)
20. Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R., Tang, J., Liu, H.: Feature selection: a data perspective. *CoRR* [arXiv:1601.07996](https://arxiv.org/abs/1601.07996) (2016)
21. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. *CoRR* [arXiv:1711.05101](https://arxiv.org/abs/1711.05101) (2017)
22. Lund, H., Østergaard, P.A.: Electric grid and heat planning scenarios with centralised and distributed sources of conventional CHP and wind generation. *Energy* **25**(4), 299–312 (2000)
23. Masci, J., Meier, U., Cireşan, D., Schmidhuber, J.: Stacked convolutional auto-encoders for hierarchical feature extraction. In: International Conference on Artificial Neural Networks, pp. 52–59. Springer, Berlin (2011)
24. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press (2012)
25. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: Neural Information Processing Systems (2017)
26. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É.: Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
27. Plaut, E.: From principal subspaces to principal components with linear autoencoders. *CoRR* [arXiv:1804.10253](https://arxiv.org/abs/1804.10253) (2018)
28. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contractive auto-encoders: explicit invariance during feature extraction. In: Proceedings of The 28th International Conference on Machine Learning (ICML-11), vol. 1, pp. 833–840 (2011)

29. Schölkopf, B., Smola, A., Müller, K.R.: Kernel principal component analysis. In: International Conference on Artificial Neural Networks, pp. 583–588 (1997)
30. Seide, F., Li, G., Chen, X., Yu, D.: Feature engineering in context-dependent deep neural networks for conversational speech transcription. In: Workshop on Automatic Speech Recognition & Understanding, pp. 24–29. IEEE (2011)
31. Serrà, J., Arcos, J.L.: An empirical evaluation of similarity measures for time series classification. CoRR [arXiv:1401.3973](https://arxiv.org/abs/1401.3973) (2014)
32. Stańczyk, U., Lakhmi, J.C. (eds.) Feature Selection for Data and Pattern Recognition, 1st edn. Springer-Verlag, Berlin (2015)
33. Van Der Maaten, L., Hinton, G.: Visualizing data using t-SNE. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008)
34. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: WaveNet: a generative model for raw audio. CoRR [arXiv:1609.03499](https://arxiv.org/abs/1609.03499) (2016)
35. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, New York (2000)
36. Wang, Q.: Kernel principal component analysis and its applications in face recognition and active shape models. CoRR [arXiv:1207.3538](https://arxiv.org/abs/1207.3538) (2012)
37. Zheng, Y., Liu, Q., Chen, E., Ge, Y., Zhao, J.L.: Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks. Lecture Notes in Computer Science, vol. 8485 (2014)

# Deep Learning Application: Load Forecasting in Big Data of Smart Grids



Abdulaziz Almalaq and Jun Jason Zhang

**Abstract** Load forecasting in smart grids is still exploratory; despite the increase of smart grids technologies and energy conservation research, many challenges remain for accurate load forecasting using big data or large-scale datasets. This chapter addresses the problem of how to improve the forecasting results of loads in smart grids, using deep learning methods that have shown significant progress in various disciplines in recent years. The deep learning methods have the potential ability to extract problem-relevant features and capture complex large-scale data distributions. Existing research in load forecasting tends to focus on finding predicted loads using small historical datasets and the behavior of the load's consumers in smart grids. Moreover, current research which applies the conventional deep learning methods for load forecasting has shown better performance than conventional load forecasting methods. However, there is little evidence that researchers have addressed the issue of hybridizing different deep learning methods for complex large-scale load forecasting in smart grids, with the intent of building a robust predictive model in smart grids and understanding the relationships that exist between different predictive models and deep learning methods. Consequently, the purpose of this chapter is to provide an overview of how the load forecasting performances using deep learning methods in smart grids can be improved.

**Keywords** Energy consumption prediction · Deep learning · Load forecasting · Smart grids

---

A. Almalaq (✉)

The Department of Electrical Engineering, University of Hail, Hail, Saudi Arabia  
e-mail: [a.almalaq@hotmail.com](mailto:a.almalaq@hotmail.com)

J. J. Zhang

The Department of Power Engineering, Wuhan University, Wuhan, China

## 1 Introduction

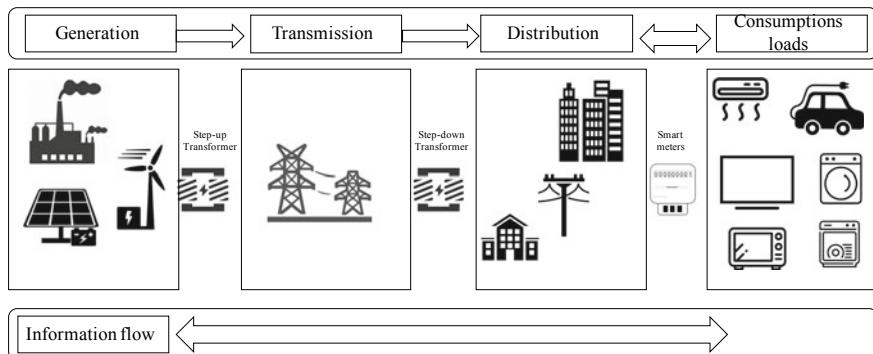
Today's power system infrastructure has been developed and improved using new technologies in different aspects. The new concept of power system grids, "smart grids", is a modern power system infrastructure that aims to build robust, reliable, efficient grids and minimize the cost of production. Enhancing the grids with renewable energy resources, automated control, and communication technologies provides possible means of efficiency, reliability, and safety for smart grids. The objective of the smart grids is to advance the use of technology and communication dramatically by investing in the bidirectional flow of power and data. The smart grid infrastructure is full of advanced sensing, communicating and computing abilities that work interoperable way in different power system parts, generation, and distribution [1]. The infrastructure scheme is illustrated in Fig. 1.

The effectiveness of smart grids relies on three primary roles that can help maintain and manage the grids as follows:

- Dynamic pricing.
- Demand-side management.
- Load forecasting.

The implications from these roles highlight the need to consider the planning and operation of the power system. The dynamic pricing provides a real-time pricing and control [1]. An application of demand-side management is the demand response that can be categorized into these three aspects [2]:

- *Peak clipping*: reducing peak loads to avoid exceeding the capacity of substations.
- *Valley filling*: promoting energy storage devices during off-peak loads.



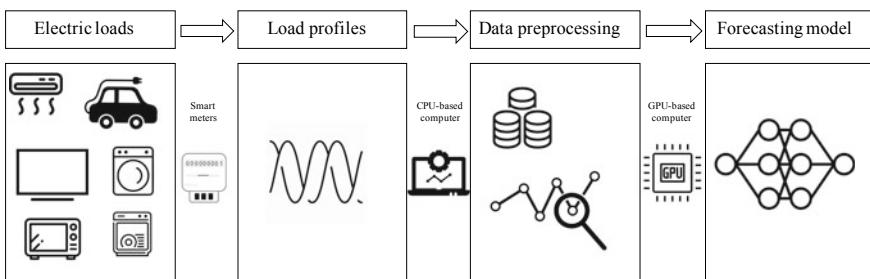
**Fig. 1** An overview of smart grids scheme. The physical part of the smart grids includes generation, transmission, distribution and electric loads. The power flow is stepped-up after the generation and stepped-down after transmission. The distribution power is measured with smart meters installed at the end-user's side. The information flow is bidirectional from the generator side to the end-user

- *Load shifting*: shifting the energy consumption, e.g., shifting the energy demand from peak load time to off-peak load time using energy storage devices.

Load forecasting is an essential task that predicts future energy consumption in order to meet the primary roles of smart grids at any time. In planning, operation, and control of the power system, load forecasting is a crucial primary element to define the distribution system capabilities that need to be obtained by the future system. If load forecasting is applied inaccurately, all relevant steps will affect the planning of future loads, and the entire planning and operation are at risk. Accurate load forecasting not only helps in optimizing the future generating units, it also saves the investment of future power facilities and helps to define the risk factors in planning, operation, and control tasks. Moreover, electricity price forecasting provides useful information for power suppliers and customers using a developed bidding system. Both suppliers and customers need accurate price predictions in order to establish their bidding strategies to maximize their profits and benefits. Therefore, to achieve the smart grids' goals, accurate and efficient load and price forecasting has become a crucial technique.

Although extensive research has been done using different physical and statistical models, accurate electric load forecasting remains a challenge in smart grids. Various artificial intelligence techniques and machine learning algorithms used in the load forecasting problem are still insufficient to predict the load in the desired form accurately. Moreover, most of these models are based on small datasets, and their prediction errors are relatively high. Enhancing the smart grids with deep learning methods to forecast loads will provide accurate predictions and efficient predictive modeling as illustrated in Fig. 2.

In this chapter, we will explore the importance of load forecasting in the energy industry and power systems; in particular, how energy consumption and electrical loads are reflecting the critical decisions in smart grids. We will research the traditional deep learning methods used for load forecasting in smart grids and investigate the hybrid deep learning methods applied for load forecasting using a real big dataset.



**Fig. 2** An overview load forecasting scheme. The aggregated electric loads are represented as load profiles using the smart meters. The CPU-based computer is used for data preprocessing e.g., data cleaning and data normalization. The GPU-based computer is used to process deep learning methods. The electric loads and load profiles are illustrated in Sect. 2. The data preprocessing and forecasting model are illustrated in Sect. 4

In the next section, we will demonstrate how load forecasting is becoming a significant contributor to energy expansion, has different objective terms of periods, is affected by various influential factors, and has significant challenges of stochastic time series. In the third section, we will review the existing research of load forecasting extensively using conventional methods, machine learning methods, and deep learning methods, highlight the insight of existing issues and narrow the gaps of existing research using significant deep learning approaches. In the fourth section, we will elaborate different promising load forecasting models using hybrid deep learning methods and compare their results with existing load forecasting approaches. In the last section, we will conclude with a summary, a balanced assessment of the contribution of load forecasting in smart grids, and a roadmap for future research directions.

The content of this chapter would be useful to researchers interested in the field of electricity market forecasting as well as graduate students who research on electrical engineering problems; especially, load forecasting and energy consumption prediction.

## 2 Background

The evolution of many smart systems such as smart grids around the world has raised new challenges and opportunities for utility providers as well as households and enterprises. Before this development, energy suppliers and integrated utilities had less financial risks and energy management adventures; in addition, the end users did not have other option but to buy electricity cost-based contracts from local providers. With all assessments, providers managed and passed all tariffs and costs to their customers.

Later, developed electricity markets faced a new challenge in the competitive markets that have allowed any energy supplier to buy electricity and natural gas. Subsequently, utility costs changed from cost-based to market-based tariffs that provide end customers multiple options for the same utility based on different rates. On the suppliers' side, this competitive market developed a variety of risks such as the fluctuation of fuel prices and electricity prices and the uncertainties of renewable energy resources. Moreover, on the end user's side, energy consumption is the main risk because of the modernization of customers' lifestyles. This factor has a massive challenge of the uncertainty of customer loads and peak demands.

With these risk factors and huge uncertainties of fuel prices, renewable energy, and demands, accurate load forecasting has become an essential technique for energy market participants such as providers and end users. In addition to these risk factors, the importance of accurate predictions is based on several other factors. For examples, addressing the electrical demand and determining the peak time are essential reasons for providers, and avoiding high electricity prices and reducing energy consumptions are crucial reasons for end customers.

## 2.1 Electrical Load

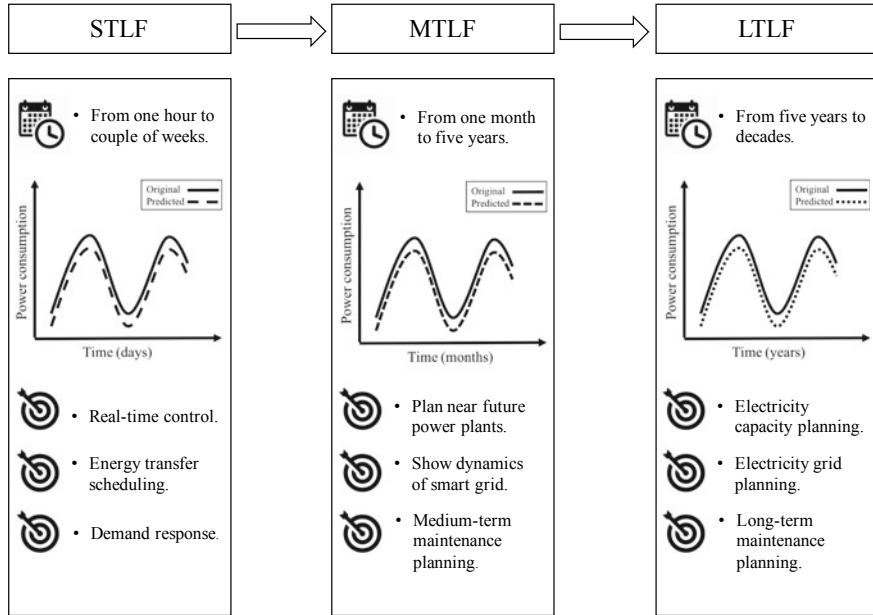
The electrical load at the distribution level is oscillatory and subject to change because human activities follow the daily, weekly and monthly event cycles. For instance, the load is generally higher in the daytime and early evening, but it is lower in the late evening and early morning. This means that every electrical appliance or light bulb that is switched on or off by customers can directly affect the electrical load seen on the distribution feeder. In general, customers buy electricity from providers to power their end-product. Therefore, the distribution system exists to deliver demand energy to customers in the form of electrical appliances and equipment, lighting, heating, and cooling as well as other demands in the commercial and industrial sectors. The distribution system of the smart grids must satisfy customer needs in order to deliver a high quality of service.

At the same time, electricity capacity is the maximum electric power generated by a specific energy resource under ideal conditions. The capacity represents the demand to have adequate resources to ensure satisfying the load peaks at all times. The generation capacity is commonly measured in kilowatts (kW) or megawatts (MW). For example, if a wind farm power plant produces 6% (2 MW) of the local generation capacity, this does not mean that it contributes to the utility with 2 MW under all conditions but it is under ideal conditions, and it is not the necessary actual amount. Indeed, the electricity generation is the amount of energy produced for a specific period of time, and it is commonly measured in kilowatt hours (kWh) or megawatt hours (MWh). For example, if the wind farm generation plant runs at its maximum capacity for three consecutive hours, the wind farm plant will produce 6 MWh of energy. If the wind farm runs at only half of its maximum capacity for these three hours, it will produce 3 MWh of energy. Generally, many energy resources do not operate at their maximum capacity all the time. Therefore, the produced energy may vary according to the conditions at the power plants.

Accordingly, the electrical load demand is a trade-off between the high quality of service and electricity generation capacity. Besides, the uncertainty of fuel, renewable energy, and actual load demands are substantial risk factors which are considered on the suppliers' side. Therefore, energy suppliers need adequate planning models and efficient forecasting models to determine the actual loads and satisfy customer demands.

## 2.2 Load Forecasting

Load forecasting is a technique usually used by energy suppliers to predict future energy consumption to meet the load demand and supply balance in the generation, transmission, and distribution markets. Household owners also use it, building managers in the commercial sector or energy supervisors in the industrial sector to meet their energy requirements and build their bidding strategies. Therefore, the load forecasting strategy is indispensable for all active energy market players. Generally,



**Fig. 3** Different types of load forecasting. The time horizons of each category is illustrated with the purpose of the application. The average time interval of the STLF is days, the MTLF is months, and the LTLF is years

The forecasting technique is used to predict load, electricity price, fossil fuels, wind power, and solar power. In this chapter, we will focus on the load forecasting in the literature review and our case study model. We will elaborate on the deep learning methods applied for load forecasting.

Different categories of load forecasting differ in the time horizon perspective. These load forecasts categorize the purpose of the prediction in the future time as illustrated in Fig. 3. We define the three main categories and their objectives as follows [3, 4]:

- *Long-term load forecasting (LTLF)*: The time interval of this type of forecasting lies from five years to decades in the future. The objective of the LTLF application is mainly for the generation and transmission systems which aim to plan for the future electricity capacity or grid by the size and cost efficient.
- *Medium-term load forecasting (MTLF)*: The forecasting time interval of this type prevails from a month to five years. The purpose of the MTLF is essentially to plan for near future power plants and show the dynamics of the smart grid.
- *Short-term load forecasting (STLF)*: This type handles time horizons of a single hour up to a couple of weeks. The STLF is necessary for the scheduling of power plants. Besides, the applications of this type of forecasting include real-time control, energy transfer scheduling, economy dispatch, and demand response.

## 2.3 Influential Factors of Load Forecasting Models

The elemental purpose of load forecasting is to predict future load patterns for cost saving and better planning and operation. The prior knowledge of the influential factors that affect the load patterns is a substantial key for accurate load predictions. The different influential factors on the electrical load and energy demand were identified, researched and utilized in different papers in the literature [5–7]. These factors are difficult to distinguish certainly due to different types of time forecasting models which may influence the STL, MTLF, and LTLF. The important factors that should be considered while modeling load forecasting, are classified as follows:

- *Time factor*: The electrical load varies with respect to customers activities. In a daily load pattern or energy consumption, it is worth noticing that the higher energy demands at certain timings. In general, the load demand is higher in the day time than the night time. For instance, industrial and commercial energy consumptions are higher at working times while residential energy consumption is higher at evening times. The working hours and working days are crucial because the variation in load patterns. The early working hours are less consumption than the middle working hours. Similarly, weekends are less energy consuming than working days. The energy consumption in holidays is more difficult to forecast because of the infrequent activities. The load curve in each time resolution such as daily, weekly, monthly or yearly is periodic but variant and inconsistent. The load curve always has the highest time of the day, the day of the week, the week of the month, and the month of the year.
- *Weather factor*: Significant weather conditions are influential factors on load forecasting. The weather conditions include temperature, humidity, wind speed, and cloud cover. These conditions can be considered mostly for the STL modeling. The high temperature in the summer season can affect the customers' comfort, and they will consume more energy for cooling. Likewise, the low temperature in the winter season can affect the customers' feeling, and they will use more energy for heating. Therefore, a strong positive correlation between high temperature and energy consumption in the summer season and a strong negative correlation between low temperature and energy consumption. The humidity is a relative weather condition to the severity feeling of high temperature and low temperature. Hence, customers increase their energy consumption during significant humidity and temperature conditions. Therefore, humidity is a considerable component for load forecasting.
- *Customer factor*: There are different kinds of customers who consume energy for different purposes such as residential, commercial and industrial customers. The energy consumption activities differ from one kind of customer to another. However, the load curves are slightly similar for the majority of one kind. The customer factor mainly depends on the size of the property, the type of property, the number of occupants, and the amount of electrical equipment. However, electrical equipment usage and energy consumption may vary from one consumer to another within one kind.

**Table 1** Different influential factors of load forecasting models

Factor	Include	Forecasting category
Time	Load patterns in: <i>minutely, hourly, daily, weekly, monthly, seasonally and yearly</i>	STLF and MTLF
Weather	Significant conditions in: <i>temperature, humidity, wind speed and cloud cover</i>	STLF and MTLF
Economy	Increase in: <i>fossil fuel price, electricity price, industrial and commercial development and population growth</i>	LTLF
Other	<i>Large social events, sport events and industrial experiments</i>	LTLF

- *Economy factor*: The economy factor plays an important role for load forecasting; especially, for the LTLF models. The economic factors include fossil fuels price, electricity price, industrial and commercial development, and population growth. The fuel prices can influence load curves by increasing the electricity price which impacts the customers' consumption. Likewise, low electricity prices increase energy consumption, hence, the load demand increase. The industrial and commercial development at a particular area increases the energy consumption as well as the increasing of population growth in a particular area.
- *Other factors*: Other factors can affect the load demands which are mainly non-periodic occasions and events that consume large energy consumption such as large social events, sports events, and industrial experiments. These types of high energy consumptions are difficult to predict resulting in a high average of prediction errors in the forecasting model.

In short, these factors may not influence each load forecasting model in the same way, but they are essential for consideration. Thus, the most critical factor is the time which directly impacts on the end customers activities. In addition, temperature and humidity are relevant influential factors for the load forecasting because of human feelings and activities that directly response to weather conditions with heating and cooling. Accordingly, Table 1 summarizes the different influential factors and their use in each load forecasting model.

### 3 Forecasting Modeling Issues in Smart Grids

In this section, we will highlight the existing issues of load forecasting, review the existing research of load forecasting extensively using conventional statistical methods, machine learning methods, and deep learning methods and narrow the gaps of existing research using significant deep learning approaches. First, we will take a look at some current general issues of load forecasting modeling. Then, we will give a short description of the most commonly encountered methods and highlight the advantages and disadvantages of each method. Finally, we will focus on deep

learning methods applied for load forecasting, demonstrate their key conceptual and algorithmic facets and narrow the gaps of existing methods. We will give an overview of their prediction results, the field of studies, locations, scale, the dataset used, the model used and the year of publication.

### ***3.1 General Issues with Load Forecasting Modeling***

In general, good load forecasting models and accurate predictions are vital elements to lead for appropriate planning, operation, and control. All load forecasting categories are difficult to be modeled over a planning period due to the many challenges and influential factors as mentioned above. Thus, the accurate prediction is still challenging due to the following difficulties:

- The large correlation with weather conditions. Sometimes, weather conditions are unpredictable, and it turns to an unexpected state.
- The large variation of energy consumption between customers due to the unpredictable events and activities. Also, the lack of using smart meters to record energy consumption efficiently.
- Single customer load forecasting is more difficult than forecasting the grid load. This difficulty exists because of the lack of large historical data for the single customer and the stochastic effects of the customer activities.
- Non-stationary time series effects of the electric load. These effects do not have a constant mean and variance.
- The high volatility of electrical load due to the change of seasonality and time factor effects. Sometimes, the same seasons are different from one year to another.

### ***3.2 Traditional Load Forecasting Models***

#### **Statistical-based models**

So far, various statistical-based techniques applied for load forecasting have been investigated, all of them with differing degrees of success. There are conventional forecasting models such as similar-day method, exponential smoothing, linear regression, multiple regression, Autoregressive Moving-Average (ARMA), and Autoregressive Integrated Moving-Average (ARIMA). Since the scope of this work is limited to deep learning-based methods, we will give a short description of some statistical-based methods below.

- *Similar-day method:* It is one of the naivest methods for load forecasting because the approach depends on searching for a similar day in the past. For instance, we search for days with similar characteristics in the historical load data and averaging them to find a forecasted day result. This method is fast and easy to get the overall

load behavior; however, it lacks the acquisition of the grid expansion and structural changes.

- *Exponential smoothing:* This approach depends on smoothing the time series through the use of the exponentially weighted moving average of the past load observations. This approach is robust and accurate; however, it lacks the accommodation of more than one seasonal pattern. This approach was applied for load forecasting in [8].
- *Linear and multiple regressions:* Regression is a statistical tool to estimate the relationship between a dependent variable and independent variables. It helps to understand how the dependent variable is related to the change of independent variables and which one is more related to the dependent variable. Linear regression is a simple regression method that accommodates one dependent variable and one independent variable and predicts the dependent variable as a function of the independent variable. It finds the best fitted straight line between the points of these variables and it is called the regression line. Multiple regression is an extension of the simple regression and it accommodates one dependent variable and more than one independent variables. The dependent variable could be the measured load data within a certain period of time and the independent variables could be any influential factor e.g., a day of the week, temperature, population size, etc. The regression approaches are easy to be modeled and calculate, however, they are sensitive to data outliers and linearity assumptions.
- *ARMA and ARIMA:* The autoregressive (AR) model and moving average (MA) model helps to understand the correlation between dependent and independent variables and predict future values of the dependent variables. The AR model uses the association between the observations and its own lagged values and the MA model uses the moving average for lagged observations and finds residual errors. The advance model ARIMA includes integrated (I) that subtracts the current observations from past observations to make the time series stationary. The models usually referred to their level of orders such as ARMA (p, q) and ARIMA (p, d, q). The lag order, which is the number of lag observations included in the model, is denoted as p. The degree of difference, which is the time of raw observations differenced, is denoted as d. The moving average order, which is the size of the window, is denoted as q. Therefore, the success of the models depends on the developer experience and skills to choose the right orders. This approach was widely used for load forecasting; especially, for the STLF in [9, 10].

### **Machine learning-based models**

On a similar note, machine learning-based techniques are widely known for their ability to accommodate complex systems, non-linear models and non-stationary time series. These advantages advance the machine learning-based models over traditional statistical-based models that must have prior influential factors, knowledge, and modeling experience to achieve accurate load forecasting. The machine learning-based techniques are self-learning methods that can classify and predict the input and output data automatically through the algorithms. Besides, there is no necessary

experience and knowledge of the forecasting model to achieve accurate load forecasting. Relevant findings of machine learning-based techniques concerning load forecasting problems in the literature revealed acceptable prediction errors; however, these attempts did not provide better performance than deep learning-based methods. Since this work concentrate on deep learning-based techniques applied for load forecasting, we will give a short overview of some machine learning-based techniques such as decision tree regression, support vector regression and artificial neural networks below.

- *Decision tree regression:* The approach builds a predictive regression model based on partitioning the data into subsets that form a tree structure. It partitions the data into smaller and smaller subsets while the decision tree increase in each fold. The tree structure has decision nodes and leaf nodes that consist of two or more branches and one numerical target, respectively. The branches in the tree represent the attribute values of the observations. The decision tree regression has continuous values of the target variables. This approach was implemented for the LTLF in [11] and energy demand modeling in buildings in [12].
- *Support vector regression:* It is a supervised learning method that represents the data observations into points in the space of the data categories. The mapped points are separated and divided by a hyperplane between the categories. Ideally, the hyperplane should be large and clear. The method was utilized to load forecasting in [13–15].
- *Artificial neural networks:* The approach is one of the widely-used techniques in machine learning. It is brain-inspired that mimics the process of human self-learning. The method architecture consists of one input layer, one hidden layer, and one output layer; however, when it has more than one hidden layer, it is considered as a deep neural network or deep learning. Generally, the connections between the artificial neurons are called edges which have connection weights of neurons. The learning process is computed by the weights and non-linear activation function. This method was widely utilized for load forecasting in [16–19, 20].

## Deep learning-based models

Advance machine learning techniques are called deep learning because they have deeper neural networks that compute more complex systems using multiple layers of non-linear functions. The advantages of deep learning models over machine learning are more complex feature extractions, less modeling, and more accurate predictions; however, its computational cost is higher than machine learning and statistical models. The top records in the accuracy of deep learning-based models were found in many important problems such as face detection, image processing, recommender systems, natural language processing, and time series predictions. Although few efforts are conducting deep learning-based models for load forecasting, for example, multilayer perceptron, convolutional neural networks, recurrent neural networks, long short-term memory, and gated recurrent unit and produced more accurate predictions, most these attempts were based on conventional implementations. Since this thesis concentrates on deep learning-based techniques applied for load forecasting, we will give a short review of some deep learning-based methods and their

applications, as well as elaborating on their key conceptual and algorithmic facets for load forecasting application.

- *Multilayer Perceptron*: Subsequent work with artificial neural networks has shown that they are capable of having multi hidden layers and performing many non-linear activation functions. This consequent work is called multilayer perceptron (MLP) that includes one input layer, more than one hidden layers, and one output layer. The approach is often used for supervised learning problems of classifications and predictions. Since it is a class of artificial neural networks, it has the same properties of the learning process, using connection weights and non-linear activation functions. The training process of the method involves weights and biases of the model to minimize the prediction error. This process consists of a forwarding pass that computes relative prediction errors and backward pass that computes gradient descent of the relative prediction errors. The backward pass is usually computed using Backpropagation algorithm that finds the partial derivatives of the activation function with respect to weights and biases. In general, the MLP method aims to self-learn the complex model and minimize prediction errors. The approach was employed in load forecasting [21] and in electricity price forecasting [22, 23]. In the context of load forecasting, a multilayer perceptron is trained on input temporal data  $X(t)$  in order to predict a target load  $L(t)$ . The  $X(t)$  can be any univariate time series or multivariate time series that includes influential factors plus historical load data. The  $L(t)$  can be a temporal shift of the univariate time series or multivariate time series. At time step  $t$ , the input layer processes the features of  $X(t) \in \mathbb{R}$ ; hence, the temporal vector is as follows:

$$X = \{X_0, X_1, \dots, X_T\} \quad (1)$$

where  $X(t)$  is the historical data at a time  $t$  and  $t \in \{1, 2, \dots, T\}$ . The load forecasting output  $L(t)$  is computed as follows:

$$L(t) = f(W \times X(t) + b), \quad (2)$$

where  $f(\cdot)$  denotes the activation function, usually implemented by a sigmoid function, a hyperbolic tangent or a rectified linear unit. The  $W$  and  $b$  are the weight matrix and the bias vector, respectively.

- *Convolutional neural networks*: Similarly, convolutional neural networks (CNN) are somewhat similar to MLP by having one input layer, more than one hidden layers, and one output layer. However, the hidden layers in this approach are convolutional layers that apply cross-correlation computation of the inputs neurons. The approach is applied widely in various applications including image recognition, video recognition, recommender systems, and natural language processing. This method is commonly used for processing grid data topology which includes a two-dimensional grid of pixels for image data construction [24]. However, the construction of the time series data is one-dimensional grid at a time interval. Thus,

load forecasting applications utilize one-dimensional CNN while the mathematical convolution operation is employed in at least one of the hidden layers [24]. This approach was used for load forecasting [25–28]. The one-dimensional convolutional neural network is described with cross-correlation function or sliding dot product as follows [24]:

$$S(t) = (X * W)(t) = \sum_{\alpha=-\infty}^{\infty} X(\alpha)W(t - \alpha), \quad (3)$$

$$L(t) = f(W_L \times S(t) + b_L), \quad (4)$$

where  $X$  denotes inputs,  $W$  is the weighting function (kernel filter),  $\alpha$  is the weighted average, and  $S$  is the convolutional output which is called feature map for the continuous time  $t$ . The  $L(t)$  denotes the load forecasting outputs,  $f(.)$  denotes the activation function, the  $W_L$  denotes the hidden to output weights and the  $b_L$  is the hidden to output bias vector.

- *Recurrent neural networks:* On the same subject, recurrent neural networks (RNN) are another special type of MLP that have one input layer, more than one hidden layer, and one output layer. However, these hidden layers have recurrent connections that make them suitable for sequential computation. The recurrent connections are from the output to the input in the hidden layer. It is commonly applied for time series sequence because it has a memory state in its architecture that assists sequential data to be processed. The approach was utilized for multiple load forecasting problems, for example in [29–31]. The mathematical representation for the RNN is defined as follows:

$$h(t) = f((W_h \times h(t - 1) + b_h) + (W_X \times X(t) + b_X)), \quad (5)$$

$$L(t) = f(W_L \times h(t) + b_L), \quad (6)$$

where  $f(.)$  denotes the activation function,  $X(t)$  denotes inputs,  $L(t)$  denotes the load forecasting outputs,  $h(t)$  denotes the hidden state,  $h(t - 1)$  denotes the previous hidden state,  $W_X$  denotes the input to hidden weights,  $W_h$  denotes the hidden to hidden weights,  $W_L$  denotes the hidden to output weights,  $b_h$  is the hidden to hidden bias vector and  $b_L$  is the hidden to output bias vector.

- *Long short-term memory:* Generally, long short-term memory (LSTM) works essentially in the same way of the RNN, but it employs more gates for the recurrent neurons called the forget gate, update gate and output gate and more internal processing unit called the cell. Each gate has a specific function in the cell. For example, the forget gate discards unwanted information from the previous state, the updated gate updates the state with new candidates, the cell filters the current state and finds the wanted and unwanted information and the output gate selects the necessary information from the cell output. This approach received attention due to its superior performance in accurately modeling. It was used widely in load forecasting in [25, 32–35]. Since the RNN method employs only one non-linear

function, the LSTM technique imposes five different non-linear functions at the same cell. In the context of load forecasting, the mathematical representation is defined as follows:

$$i_t = g_1(W_{i,n} \times X(t) + W_{i,m} \times L(t-1) + b_i), \quad (7)$$

$$f_t = g_1(W_{f,n} \times X(t) + W_{f,m} \times L(t-1) + b_f), \quad (8)$$

$$o_t = g_1(W_{o,n} \times X(t) + W_{o,m} \times L(t-1) + b_o), \quad (9)$$

$$U = g_2(W_{U,n} \times X(t) + W_{U,m} \times L(t-1) + b_U), \quad (10)$$

$$C(t) = f_t \times C(t-1) + i_t \times U, \quad (11)$$

$$L(t) = o_t \times g_2(C(t)), \quad (12)$$

where  $g_1$  denotes the sigmoid activation function,  $g_2$  denotes the hyperbolic tangents activation function,  $X(t)$  is the input vector,  $i_t$  is the input of the input gate where the subscript means input,  $f_t$  is the input of the forget gate where the subscript means forget,  $o_t$  is the input of the output gate where the subscript means output,  $U$  is the update signal,  $C(t)$  is the state value at the time  $t$  of computation and  $L(t)$  is the output of the cell for load forecasting.  $W_{(.)}$  and  $b_{(.)}$  are the weight matrices and bias vectors, respectively. The weights correspond to the current state values of a particular variable are denoted as  $W_{(,),n}$  and previous state signal as  $W_{(,),m}$ .

- *Gated recurrent unit*: Similarly, gated recurrent unit (GRU) is another recent and popular gated architecture of the RNN that adaptively captures dependencies and features of time series. It also solves the problem of vanishing gradient descent. The main difference between this approach and LSTM is that it has a single update gate and a reset gate. The update gate  $z_t$  combines the forget gate and the input gate of the LSTM method to control the unwanted and wanted information. The reset gate  $r_t$  reconstructs the cell memory with the next processed input. This approach outperformed the LSTM in [36]. Also, it was utilized for load forecasting problems in [37, 38]. The mathematical representation for load forecasting context is defined as follows:

$$z_t = g_1(W_{z,n} \times X(t) + W_{z,m} \times L(t-1) + b_z), \quad (13)$$

$$r_t = g_1(W_{r,n} \times X(t) + W_{r,m} \times L(t-1) + b_r), \quad (14)$$

$$U = g_2(W_{U,n} \times X(t) + W_{U,m} \times [r_t \odot L(t-1)] + b_U), \quad (16)$$

$$L(t) = (1 - z_t) \odot h(t - 1) + z_t \odot U, \quad (17)$$

where  $g_1$  denotes the sigmoid activation function,  $g_2$  denotes the hyperbolic tangents activation function,  $X(t)$  is the input vector,  $L(t)$  is the output vector of load forecasting,  $U$  is the update signal, and  $\odot$  is element-wise multiplication.  $W_{(.)}$  and  $b_{(.)}$  are the weights' matrices and bias vectors, respectively. The weights correspond to the current state values of a particular variable are denoted as  $W_{(.),n}$  and previous state signal as  $W_{(.),m}$ .

Since these approaches are subcategories of the RNN, they are appropriate tools for sequential problems such as time series prediction and load forecasting. Additionally, they solve the problem of vanishing gradient descent in the RNN by avoiding any bias of recent observations.

### 3.3 Traditional Load Forecasting Models

Modeling deep learning-based paradigm for load forecasting is not an easy task. There are a large number of choices and parameters that have to be adequately made to achieve the appropriate modeling and accurate predictions. However, few guidelines can help developers to overcome these challenges. We will go over these guidelines in the next section, but now we classify these challenges as the following:

- *Data scale*: The data scale of the historical load is a major influential factor that affects the deep learning-based modeling. This factor can influence the model predictions because of any of the following:
  - *Outliers and missing values*: If the data scale is small, even few outliers or few missing values will form a significant alteration to the model.
  - *Train and test data*: To evaluate the model properly, the model splits the data into train and test data. Each data has enough portion of data observations to perform the proper training and testing. However, if the original data scale is small, each data may not have enough observations to perform properly.
- *Data preprocessing*: Usually, the data preprocessing is an important step that has to be conducted before the data is ready as an input to the deep learning-based model. This step helps to manage the forecasting model problems and avoid excessive volatility of the data.
- *Designing the deep learning-based model*: Selecting an appropriate deep learning method is the first step in designing an adequate model. There are many architectures of deep learning techniques that were utilized for time series predictions and electricity load forecasting.
- *The appropriate number of hidden layers and neurons*: Determining the size of hidden layers and neurons may be the most challenging task in designing deep learning-based models. This issue arises because the model needs to be fit and

have less computational cost. The small size of hidden layers and neurons may lead the model to inflexible performance with data. On the other hand, the large number of hidden layers and neurons may increase the chance of overfitting the model with the data. Besides, the larger size of the hidden layers and neurons is more computational complexity.

- *Model overfitting and validation:* Overfitting issue arises when the model learns the details of the training data well, however, it lacks its performance when it is tested with new data which is the unseen testing data. Although examining the model for overfitting is a good strategy for determining the excellent forecasting model, validating the model using other tests is necessary to achieve a sufficient deep learning-based model for load forecasting.
- *Offline modeling:* Generally, deep learning-based models are designed for offline training and testing. This technique learns the entire data at once and evaluates the model with a portion of the data that is testing data. On the other hand, online modeling is a dynamic model that learns at each time step of the brand-new data and update the predictive model according to the latest data.

## 4 Solutions and Recommendations

Since most of the previous research in load forecasting focuses on small historical datasets and uses conventional modeling approaches, there is little emphasis on using big data, hybridizing different significant deep learning-based models and finding optimal deep learning parameters by using different solutions such as evolutionary computation algorithms. An initial analysis was able to find evidence of using deep learning methods that are powerful techniques for precise time series predictions. We hypothesize that hybridizing two or more deep learning-based methods for load forecasting in smart grids could produce more accurate prediction and form the groundwork for explicitly broad load forecasting models. Besides, finding optimal deep learning parameters using different evolutionary computation methods could form the preliminary search space of deep learning parameters.

In this section, we will present some guidelines and solutions for the issues of modeling deep learning-based paradigms. We will elaborate a case study of different promising load forecasting models using hybrid deep learning methods and compare their results with existing load forecasting approaches.

### 4.1 Guidelines and Solutions to Modeling Issues

Since the deep learning-based models are sensitive to a large number of choices and parameters, finding the appropriate model is the point at issue. Therefore, an element key in finding an adequate model is to follow some of the following guidelines

and successful solutions that have been utilized in the literature. Few guidelines and recommendations can help model designers to achieve their goal quickly, but not every guideline will work efficiently with each load forecasting model. These guidelines and recommendations are listed according to the modeling issues that we mentioned earlier:

- *Data scale*: Nowadays because of big data of electrical loads and prices in smart grids, load forecasting designers need new approaches and technologies in order to achieve their goals. In general, the big data or a large-scale dataset has a large volume of information and complex data structures that cannot be processed with traditional load forecasting models. The large-scale dataset helps utility providers and energy management operators to analyze their systems comprehensively. Besides, they can design their forecasting models with high computational techniques such as deep learning models that perform well with big data by using batches for training. The primary objective of this chapter is to design a deep learning-based forecasting model using a large-scale dataset.
- *Data preprocessing*: Data preprocessing refers to all processing techniques on the raw data before it is fed to the deep learning model. Some preprocessing techniques are necessary to be applied before the model learns the dataset. These preprocessing techniques help the model perform better and consume less computation time. We list some of the common data preprocessing techniques used for deep learning methods below:
  - *Data cleaning*: Since deep learning-based models are sensitive to defective samples in the dataset, so data cleaning technique is essential for better deep learning performance. The technique may include removing or fixing missing data and outliers.
  - *Data normalization*: Normalizing the dataset features avoids the problem of dominating the large number ranges of attributes and helps the model to perform accurately. While most of the electrical load datasets consist of different value scales and various quantities, for example, load profiles, weather data, and fuel prices, normalizing these values before feeding to the deep learning model provides easier learning and less computation cost. The mathematical representation of data normalization is as follows:

$$X'(t) = \frac{X(t) - \min}{\max - \min} \quad (17)$$

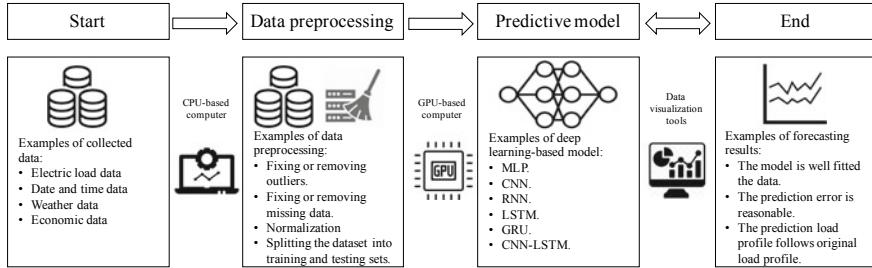
where  $X(t)$  is the original value of the input dataset,  $X'(t)$  is the normalized value scaled to the range  $[0, 1]$ ,  $\max$  is the maximum value of the features, and  $\min$  is the minimum value of the features.

- *Designing the deep learning-based model*: Selecting an appropriate deep learning architecture is the first step for the load forecasting model. Since various research papers applied deep learning methods for load forecasting, reviewing these papers, analyzing their techniques and comparing their results is an important step in

finding the best technique. Sometimes a successful technique performs well in a particular model but does not perform in the same way in another dataset. Thus, finding the best model is a challenging task for load forecasting modelers. In most of the reviewed papers, the authors made their choices based on their trials and empirical tests.

- *The appropriate number of hidden layers and neurons:* In general, the number of hidden layers is less complicated and influential in the deep learning model performance than the number of hidden neurons in the hidden layer. A single hidden layer sums the input weights and multiplies them by a non-linear activation function. An extra hidden layer can smooth and approximate the mapping features of the previously hidden layer resulting in a better prediction output, but not in every case. The number of hidden neurons is more influential on the final prediction output than the number of hidden layers. A large number of neurons leads to an overfitting problem, and a small number of neurons leads to an under fitting problem. There is no rule of choosing the number of hidden layers and the number of hidden neurons, but there are some techniques such as trial and error, pruning techniques, and evolutionary computation algorithms. The ability of evolutionary computation algorithms is that they can evolve the deep learning-based model and optimize the number of hidden neurons, for example, genetic algorithms in [39, 40].
- *Model overfitting and validation:* To improve the model validation and avoid the overfitting problem, applying a cross-validation technique is an essential task after modeling. The cross-validation splits the datasets into k-fold subsets to estimate the general performance of the prediction model and gives an insight on how the model generalizes the independent variables throughout the datasets. The method repeats the process of splitting the dataset into training and testing portions for k-times where the size of the testing data portion remains fixed but moves through the original dataset; the remainder used as a training dataset every fold.
- *Online modeling:* Applying a parallel computational technique, such as MapReduce, can reduce the time consumption of the deep learning-based model using one of the computational frameworks, for example, Apache Hadoop and Apache Spark. Also, applying the parallel computation technique to the proposed model can provide a real-time prediction paradigm, for example, real-time power forecasting, that can train the historical inputs variables offline and update and test the current input variables online.

To model load forecasting efficiently using big data and deep learning methods, four crucial steps have to be implemented to obtain an accurate prediction. These steps start with determining the big data that may include load data and other influential factors. Then, the data preprocessing consists of the data cleaning, normalization and splitting the data into training and testing sets. Selecting a deep learning method that may be suitable for the forecasting problem and designing the deep learning model properly is the third step. Finally, the forecasting results may be visualized using visualization tools to evaluate the model performance and prediction. If the



**Fig. 4** An overview of deep learning-based model procedure. The procedure consists of four main segments. The first segment is selecting an electricity load dataset which may include influential factors data. The second segment is the data preprocessing using a CPU-based computer. This means that the preprocessing does not need high computation processing. The third segment is the predictive model which is one of the deep learning algorithms. This step need a high computational tool such as a GPU-based computer. The last step shows the forecasting results and prediction errors. This step needs visualization tools to visualize the outputs such as prediction graphs, training and testing performances, and comparative charts

prediction errors are not reasonable, the deep learning model could be modified or changed to improve the prediction accuracy. Besides, the model can be compared with other deep learning methods. Figure 4 demonstrates the general modeling procedures of load forecasting.

## 4.2 Case Study

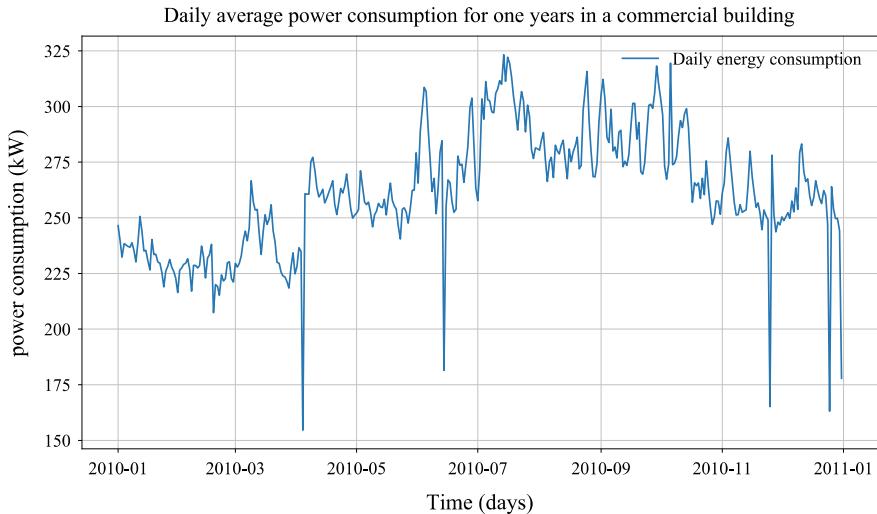
In this case study, we consider a big dataset in the form of power consumption in a commercial building. First, we perform some analysis and preprocessing techniques in order to understand the nature of the time series dataset and make it ready for the forecasting model. Then, we set up a hybrid deep learning-based model for STLF in an hour-ahead, a day-ahead and a week-ahead forecasting. We compare the forecasting results with traditional statistical-based models, machine learning-based models and deep learning-based models.

### Commercial building data

The large-scale dataset of power consumption in a commercial building is publicly published in [41]. The time series dataset consists of one year in 2010 with fifteen minutes' resolution. The dataset includes the power consumption in (kW) and outdoor temperature in (F). The chosen building in this study is building 1 which is a retail building in Fremont, CA. Figure 5 shows the variation of average power consumption for 2010.

### Hybrid deep learning-based models

Referring to the modeling procedure which is shown in Fig. 7 our modeling stands for four main parts including preprocessing and hybrid deep learning-based model.

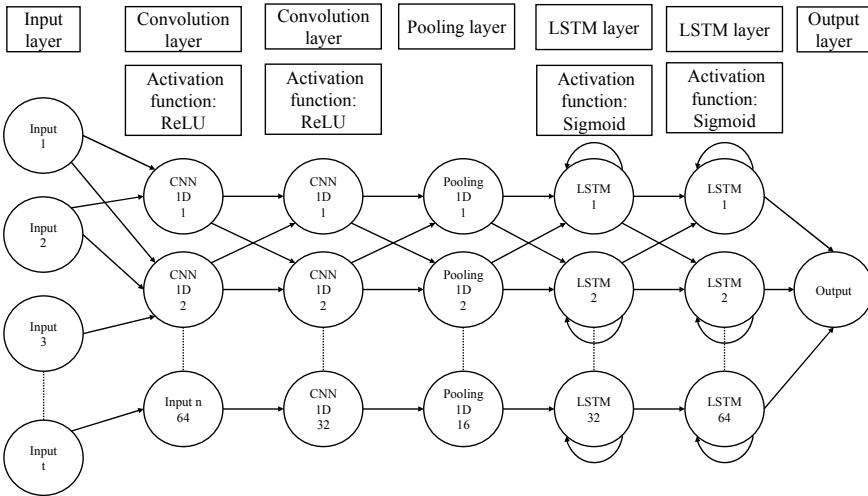


**Fig. 5** The load profile in kilowatts (kW) of the averaged daily power consumption of a commercial building for one year

The data preprocessing segment prepares the input features collected in the dataset to the hybrid deep learning-based model.

There are three main steps in the preprocessing segment where the first depends on normalizing the original datasets as in (17), the second is preparing the input data for the supervised learning technique, and the third is splitting the normalized supervised dataset into three parts, the training, the validating and the testing datasets. To evaluate the performance of the proposed model accurately, the training data is used for the training process of the approach, the validating data is used to validate the model performance, and the testing data is used just for testing the forecasting process using unseen data.

The hybrid deep learning-based model in the third step is based on a coder and decoder which are the CNN model and LSTM model, respectively. The input of the CNN-LSTM is the record of power consumption dataset of the commercial building after the preprocessing analysis, and the output is the power consumption forecasting for the next day and next week. It is unlike traditional CNN or LSTM models because hybridizes these two superior methods to improve the learning process. The first half is CNN, which is utilized to extract the input features and encode them as in (5) and (6), and the second half is the LSTM, which is used to analyze the extracted features as in (7)–(12) from the CNN and decode the features to predict the power consumption for the next period of time. The approach includes two layers of the one-dimensional CNN to improve extracting the input features, one layer of the one-dimensional pooling to collect the extracted features, and two layers of the LSTM to analyze the collected extracted features and predict the output as shown in Fig. 6.



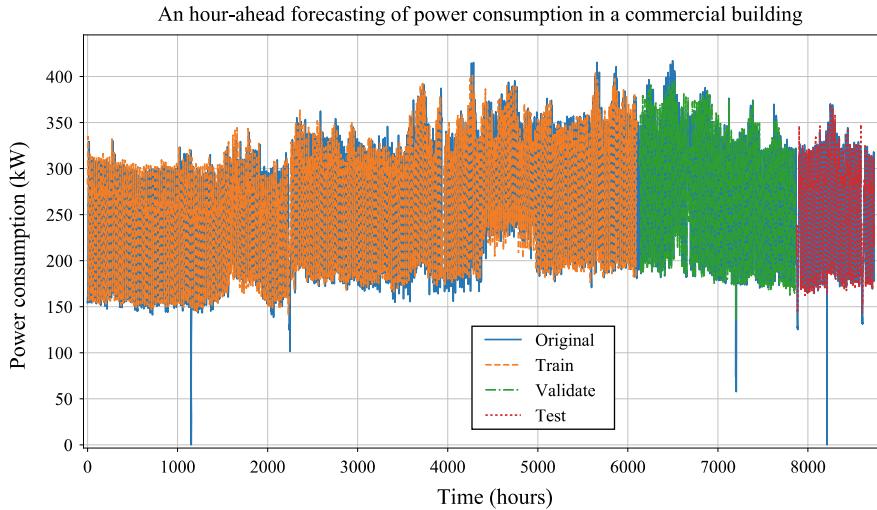
**Fig. 6** The architecture of the hybrid deep learning-based model. Circles in the input layer and output layer represent input  $X(t)$  and output  $L(t)$ , respectively. Circles in the other layers represent the cells of each layer. The activation function in the convolutional layers is ReLU and in the LSTM layers is sigmoid

This CNN-LSTM model is implemented using Python 2.7, the Keras deep learning framework [42], and the scikit-learn framework [43]. We configured the model network with the same parameters and activation functions shown in Fig. 6. Because the CNN model has a few choices of the number of neurons, we selected the number of hidden neurons as 64 neurons in the first convolution layer, 32 neurons in the second convolution layer and 16 neurons for the pooling layer. Thus, the decoder segment reverses the number of hidden neurons with 32 and 64 hidden neurons. The applied optimizer function is Adam, and the applied loss function is the mean square error. The total number of training epochs is 1000.

## Results and discussions

To evaluate the forecasting performance results, we utilized 70% from the original datasets to train the approach model, 20% from the original dataset to validate the performance of the model and the last unseen 10% from the original dataset to test the model predictions. The conventional metrics used to evaluate the predictive models are utilized to evaluate the forecasting in our experiments. The traditional metrics such as the root-mean-squared error (RMSE), and the coefficient of variation of the RMSE, known as the normalized root-mean-squared error (NRMSE), and mean absolute percentage error (MAPE), are defined as follows:

$$RMSE = \sqrt{\sum_{t=1}^T (L(t) - X(t))^2}, \quad (18)$$



**Fig. 7** The energy consumption forecasting graph results of the CNN-LSTM model. The forecasting curves of one hour-ahead are represented in dashed lines that follow the original load profile line curves

$$NRMSE = \frac{RMSE}{\bar{X}} \times 100\%, \quad (19)$$

$$MAPE = \frac{1}{T} \sum_{t=1}^T \frac{|X(t) - L(t)|}{|X(t)|} \times 100\%, \quad (20)$$

where  $T$  represents the total number of time steps in the time series dataset,  $L(t)$  is the predicted output of the time series,  $X(t)$  is the real measured time series in the dataset, and  $\bar{X}$  is the average of the actual values of power consumption.

As shown Fig. 7, the forecasting result is in dashed line curves with triangles and the original data is shown with line curves. It is worth noticing, the forecasting curves are almost consistent with the original curves except for several abrupt deviation points. This represents the effectiveness of the CNN-LSTM forecasting model.

Applying the cross-validation method to the CNN-LSTM model produces a robust averaged estimation of the forecasting when each observation in the dataset is used for training and testing at each fold. We utilized 10-fold cross-validation in our forecasting model using a time series cross-validator [43]. By applying this method, we avoided the overfitting problem in our model and validated the prediction model by testing unseen data at each fold. Besides, we compared our model with traditional statistical, machine learning and deep learning models as in Table 2. It is worth noticing that the best forecasting model performance was the CNN-LSTM for the one hour-ahead forecasting and one day-ahead forecasting. Also, the one-dimensional CNN model performed better than other models. The LSTM and GRU models performed in a similar way for both time prediction resolutions. The GRU performance was a little bit better than the LSTM in both forecasting steps. The decision tree model was

**Table 2** Comparison of an hour-ahead forecasting and a day-ahead forecasting between traditional models and hybrid deep learning-based model. The prediction errors are in the percentage of NRMSE and MAPE and represent the average of cross-validation error

Model	An hour-ahead forecasting		A day-head forecasting	
	NRMSE (%)	MAPE (%)	NRMSE (%)	MAPE (%)
ARIMA	9.979	7.680	6.250	4.644
Decision tree	14.410	12.176	7.141	5.054
KNN	11.242	8.549	6.649	4.465
MLP	9.017	7.373	6.597	3.966
CNN	7.559	5.570	5.893	4.108
LSTM	9.002	7.213	6.388	4.719
GRU	8.969	7.011	6.500	4.673
<b>CNN-LSTM</b>	<b>6.672</b>	<b>4.096</b>	<b>5.211</b>	<b>3.693</b>

the worst forecasting performance in our case study. Therefore, the CNN-LSTM model showed its superiority of forecasting since it is hybridizing two successful deep learning-based models.

## 5 Conclusions and Future Trends

The infrastructure of the energy market has changed dramatically in recent years. With the development of the smart technologies implemented in the grid, the introduction of renewable energy resources and distributed energy resources, energy market participants are in need to update their methodologies for planning, operating, and controlling electrical loads and energy consumptions. This chapters focused on the deep learning application applied to load forecasting in smart grids; thus we gave a snapshot of the background of smart grids and electrical load patterns. We discussed the importance of the load forecasting in the energy market and the factors influencing the load forecasting modeling.

In this overview, we reviewed traditional load forecasting methods such as statistical methods, machine learning methods, and deep learning methods. We explored the key conceptual and algorithmic facets of deep learning methods applied to load forecasting, and discussed the general issues of deep learning modeling. Also, We performed a case study of big data and hybrid deep learning-based model for a commercial building load forecasting. We found the CNN-LSTM model outperformed other traditional deep learning models.

From the literature review, we conclude that no specific deep learning model outperforms other deep learning models for every forecasting problem. Thus, the best architecture choice depends on the forecasting task and challenges. The LSTM and GRU models are close to each other in their performances because they are subcategories of RNN and suitable for sequential problems. They usually accomplish

accurate predictions compared with traditional models. As a suggestion, hybridizing two superior models from the literature is an excellent technique to achieve a good performance in load forecasting modeling as our results in the case study show.

Reinforcement learning is a possible future research subject of load forecasting and energy management. This method can allow the energy model to adjust the parameters and reduce energy consumption automatically. Furthermore, by integrating big data that includes many electrical load features, not only traditional aggregated load and energy consumption can be forecasted, but a comprehensive interactive prediction of many energy systems such as operational energy parameters and electric vehicle charging can be achieved. Besides, the implementation of intelligent predictive systems in actual systems can increase the smart grid development and enhance practical future intelligent applications.

## References

1. Gungor, V.C., et al.: Smart grid technologies: communication technologies and standards. *IEEE Trans. Ind. Inf.* **7**(4), 529–539 (2011)
2. Deng, R., Yang, Z., Chow, M., Chen, J.: A survey on demand response in smart grids: mathematical models and approaches. *IEEE Trans. Ind. Inf.* **11**(3), 570–582 (2015)
3. Almalaq, A., Edwards, G.: A review of deep learning methods applied on load forecasting. In: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 511–516 (2017)
4. Raza, M.Q., Khosravi, A.: A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings. *Renew. Sustain. Energy Rev.* **50**, 1352–1372 (2015)
5. Khatoon, S., Ibraheem, Singh, A.K., Priti: Effects of various factors on electric load forecasting: an overview. In: 2014 6th IEEE Power India International Conference (PIICON), pp. 1–5 (2014)
6. Fahad, M.U., Arbab, N.: Factor affecting short term load forecasting. *J. Clean Energy Technol.* **2**(4), 305–309 (2014)
7. Feinberg, E.A., Genethliou, D.: Load Forecasting. In: Chow, J.H., Wu, F.F., Momoh, J. (eds.) *Applied Mathematics for Restructured Electric Power Systems: Optimization, Control, and Computational Intelligence*, pp. 269–285. Springer US, Boston, MA (2005)
8. Ji, P., Xiong, D., Wang, P., Chen, J.: A study on exponential smoothing model for load forecasting. In: 2012 Asia-Pacific Power and Energy Engineering Conference, pp. 1–4 (2012)
9. Amjady, N.: Short-term hourly load forecasting using time-series modeling with peak load estimation capability. *IEEE Trans. Power Syst.* **16**(3), 498–505 (2001)
10. Hagan, M.T., Behr, S.M.: The time series approach to short term load forecasting. *IEEE Trans. Power Syst.* **2**(3), 785–791 (1987)
11. Ding, Q.: Long-term load forecast using decision tree method. In: 2006 IEEE PES Power Systems Conference and Exposition, pp. 1541–1543 (2006)
12. Yu, Z., Haghhighat, F., Fung, B.C.M., Yoshino, H.: A decision tree method for building energy demand modeling. *Energy Build.* **42**(10), 1637–1646 (2010)
13. Chen, B.-J., Chang, M.-W., et al.: Load forecasting using support vector machines: a study on EUNITE competition 2001. *IEEE Trans. Power Syst.* **19**(4), 1821–1830 (2004)
14. Pai, P.-F., Hong, W.-C.: Support vector machines with simulated annealing algorithms in electricity load forecasting. *Energy Convers. Manag.* **46**(17), 2669–2688 (2005)
15. Zhu, Z., Sun, Y., Li, H.: Hybrid of EMD and SVMs for short-term load forecasting. In: 2007. ICCA 2007. IEEE International Conference on Control and Automation, pp. 1044–1047 (2007)
16. Park, D.C., El-Sharkawi, M.A., Marks, R.J., Atlas, L.E., Damborg, M.J.: Electric load forecasting using an artificial neural network. *IEEE Trans. Power Syst.* **6**(2), 442–449 (1991)

17. Hayati, M., Shirvany, Y.: Artificial neural network approach for short term load forecasting for Illam region. *World Acad. Sci. Eng. Technol.* **28**, 280–284 (2007)
18. Kandil, N., Wamkeue, R., Saad, M., Georges, S.: An efficient approach for short term load forecasting using artificial neural networks. *Int. J. Electr. Power Energy Syst.* **28**(8), 525–530 (2006)
19. Zhang, G., Patuwo, B.E., Hu, M.Y.: Forecasting with artificial neural networks: the state of the art. *Int. J. Forecast.* **14**(1), 35–62 (1998)
20. González, P.A., Zamarreño, J.M.: Prediction of hourly energy consumption in buildings based on a feedback artificial neural network. *Energy Build.* **37**(6), 595–601 (2005)
21. Tsakoumis, A.C., Vladov, S.S., Mladenov, V.M.: Electric load forecasting with multilayer perceptron and Elman neural network. In: 6th Seminar on Neural Network Applications in Electrical Engineering, pp. 87–90 (2002)
22. Dudek, G.: Multilayer perceptron for GEFCom2014 probabilistic electricity price forecasting. *Int. J. Forecast.* **32**(3), 1057–1060 (2016)
23. Kuo, P.-H., Huang, C.-J.: An electricity price forecasting model by hybrid structured deep neural networks. *Sustainability* **10**(4), 1280 (2018)
24. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
25. Amarasinghe, K., Marino, D.L., Manic, M.: Deep neural networks for energy load forecasting. In: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), pp. 1483–1488 (2017)
26. Khan, S., Javaid, N., Chand, A., Khan, A.B.M., Rashid, F., Afridi, I.U.: Electricity load forecasting for each day of week using deep CNN. In: Kalbitzer, U., Jack, K.M. (eds.) Primate Life Histories, Sex Roles, and Adaptability, pp. 1107–1119. Springer International Publishing, Cham (2019)
27. Kollia, I., Kollias, S.: A deep learning approach for load demand forecasting of power systems. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, pp. 912–919 (2018)
28. Dong, X., Qian, L., Huang, L.: A CNN based bagging learning approach to short-term load forecasting in smart grid. In: 2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (Smart-World/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 1–6 (2017)
29. Shi, H., Xu, M., Li, R.: Deep learning for household load forecasting—a novel pooling deep RNN. *IEEE Trans. Smart Grid* **9**(5), 5271–5280 (2018)
30. Yu, Z., Niu, Z., Tang, W., Wu, Q.: Deep learning for daily peak load forecasting—a novel gated recurrent neural network combining dynamic time warping. *IEEE Access* **7**, 17184–17194 (2019)
31. Bedi, J., Toshniwal, D.: Deep learning framework to forecast electricity demand. *Appl. Energy* **238**, 1312–1326 (2019)
32. Kong, W., Dong, Z.Y., Hill, D.J., Luo, F., Xu, Y.: Short-Term residential load forecasting based on resident behaviour learning. *IEEE Trans. Power Syst.* **33**(1), 1087–1088 (2018)
33. Marino, D.L., Amarasinghe, K., Manic, M.: Building energy load forecasting using deep neural networks. In: IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society, pp. 7046–7051 (2016)
34. Gan, D., Wang, Y., Zhang, N., Zhu, W.: Enhancing short-term probabilistic residential load forecasting with quantile long–short-term memory. *J. Eng.* **2017**(14), 2622–2627 (2017)
35. Zheng, J., Xu, C., Zhang, Z., Li, X.: Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network. In: 2017 51st Annual Conference on Information Sciences and Systems (CISS), pp. 1–6 (2017)
36. Chung, J., Gülcöhre, Ç., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: CoRR (2014). <http://arxiv.org/abs/1412.3555>
37. Kumar, S., Hussain, L., Banarjee, S., Reza, M.: Energy load forecasting using deep learning approach-LSTM and GRU in spark cluster. In: 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT), pp. 1–4 (2018)

38. Gao, X., Li, X., Zhao, B., Ji, W., Jing, X., He, Y.: Short-term electricity load forecasting model based on EMD-GRU with feature selection. *Energies* **12**(6), 1140 (2019)
39. Almalaq, A., Zhang, J.J.: Evolutionary deep learning-based energy consumption prediction for buildings. *IEEE Access* **7**, 1520–1531 (2019)
40. Bouktif, S., Fiaz, A., Ouni, A., Serhani, M.A.: Optimal deep learning LSTM model for electric load forecasting using feature selection and genetic algorithm: comparison with machine learning approaches. *Energies* **11**(7) (2018)
41. Long-Term Energy Consumption & Outdoor Air Temperature For 11 Commercial Buildings-Openei Datasets. Openei.org (2019)
42. Chollet, F. et al.: Keras. GitHub (2015)
43. Pedregosa, F., et al.: Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)

# Fast and Accurate Seismic Tomography via Deep Learning



Mauricio Araya-Polo, Amir Adler, Stuart Farris and Joseph Jennings

**Abstract** This chapter presents a novel convolutional neural network (CNN)-based approach to seismic tomography, which is widely used in velocity model building (VMB). VMB is a key step in geophysical exploration where a model of the subsurface is needed, such as in hydrocarbon exploration for the Oil & Gas industry. The VMB main product is an initial model of the subsurface that is subsequently used in seismic imaging and interpretation workflows. Existing solutions rely on numerical solutions of wave equations, and requires highly demanding computation and the resources of domain experts. In contrast, we propose and implement a novel 3D CNN solution that bypasses these demanding steps, directly producing an accurate subsurface model from recorded seismic data. The resulting predictive model maps relationships between the data space and the final earth model space. The subsurface models are reconstructed within seconds, namely, orders of magnitude faster than existing solutions. Reconstructed models are free of human biases since no initial model or numerical technique tuning is required. This chapter is a significant extension of previous published material and provides a detailed explanation of the seismic tomography problem, and of the previously unpublished 3D CNN architecture, training workflows and comparisons to state-of-the-art.

**Keywords** Deep learning · Exploration geophysics · Seismic imaging · Tomography · Inverse problems · Convolutional neural network

---

M. Araya-Polo · S. Farris · J. Jennings

Data Science and Machine Learning R&D, Shell International Exploration & Production Inc., Houston, TX, USA

A. Adler (✉)

Center for Brains, Minds and Machines (CBMM), MIT, Cambridge, MA, USA  
e-mail: [adleram@mit.edu](mailto:adleram@mit.edu)

S. Farris · J. Jennings

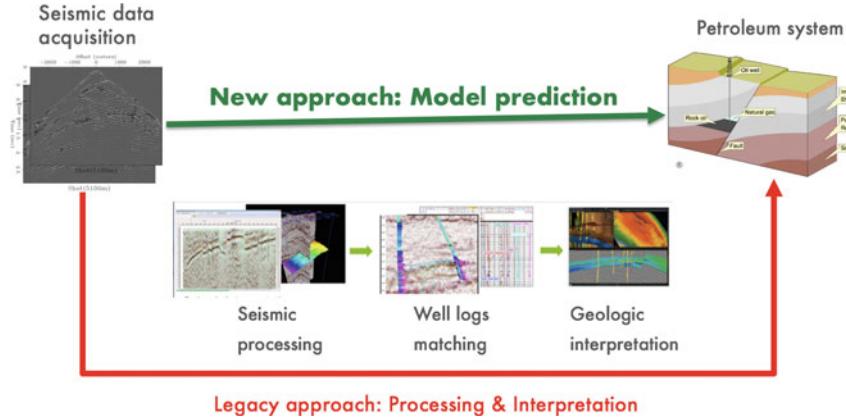
Department of Geophysics, Stanford University, Stanford, CA, USA

## 1 Introduction

The main workflow of hydrocarbon exploration starts with acquiring field data, which consist of recordings of the response of the subsurface to artificial perturbations. Following data acquisition, several disciplines [1], geology, geophysics, petrophysics, etc., combine efforts to produce a model of the earth (see Fig. 1, top right) which may or may not have clear indications of hydrocarbon presence. In areas such as the Gulf of Mexico, hydrocarbons tend to accumulate near salt bodies making them a key geological structure in earth model building [2]. This earth model is a critical part of the decision making process and is given utmost importance during exploration projects. The average success ratio of the industry is low, thus avoiding unnecessary expenses, such as drilling wells, translates into saving millions of dollars. Therefore, techniques to accelerate the decision time and increase the success ratio are crucial.

What we propose in this chapter goes beyond what is currently making inroads in exploration geosciences, which is machine learning (ML) techniques being applied to specific well-known steps of the standard hydrocarbon exploration workflow (Fig. 1, red arrow). Most of the advances happen on the interpretation [3, 4] of the models rather than in the generation of them. Alternatively, our method is a end-to-end solution, producing earth models directly from unmanipulated seismic data. Our method differs from current velocity building methods, seismic tomography [5] (similar to medical tomography but the penetrating wave is seismic) or wave equation-based modeling/inversion, in that our method is automatic and without human intervention. The deep learning (DL) technique employed follows recent work [6, 7] that demonstrates this new approach, which uses a deep neural network (DNN) statistical model to transform raw input seismic data directly to the final mapping in 2D or 3D model space. The computational cost of the proposed approach is mostly due to the training phase, which occurs only once and offline. After training, velocity model reconstruction computational costs are negligible, thus making the overall computing requirements a fraction of those needed for traditional techniques, in particular the ones involving partial differential equations (PDE)-based simulations. As a preliminary step, velocity semblance [8] is used as the input feature space, which apparent seismic velocity (main attribute of an earth model) information for the training process. While we do perform feature extraction, rather than use the raw data, this feature extraction step is automated and not subject to human bias. Later, we extend the approach to work directly on the raw recordings thus freeing it from feature extraction and using the fully accepted unmanipulated seismic data as input.

The main design concern relates to the generalization capability of the DL-based solution, which basically indicates how much a trained model can accurately predict unseen data. To address that concern, we foresee models being trained with specific data belonging to different major exploration areas such as: pre-salt (Brasil offshore) or subsalt (Gulf of Mexico or West Africa). Regarding future hydrocarbon exploration workflows, one can imagine this technique being used just after data acquisition (field recording), then trained models are loaded up to the cloud from which interpreters can access realizations, thus performing online scenarios testing when feeding back



**Fig. 1** Overall vision of the new exploration geophysics workflow (green arrow), where the classical way of approaching the problem is depicted in bottom following the red arrow

their model modifications. This anticipated workflow is fully ML-based, flexible and with the domain experts at the center of the critical decision making process. Finally, we envision that this technique can also be applied to other tomography problems that arise in the geosciences such as global seismology, shallow hazards, etc.

This chapter is organized as follows: Sect. 2 introduces the seismic tomography problem and the principles of seismic data acquisition. Section 3 explains the DL approach and the semblance geophysical feature. Section 4 presents experimental results with 2D synthetic seismic data. Section 5 compares our DL results against the state-of-the-art results obtained with industry's tool of choice. Section 6 introduces our preliminary results without extracting features from the data. Finally, conclusions and future research are provided in Sect. 7.

## 2 The Seismic Tomography Problem

To provide a complete context of the earth model building problem, before delving into our proposed DL-based solutions, this section explains the data to be used through the chapter and, in a succinct manner, reviews the scientific problem at hand.

### 2.1 Seismic Data

Seismic data are acquired, for the onshore case, via sources positioned on the earth's surface and arrays of receivers (geophones). In the offshore case, the sources and receivers (hydrophones) are towed by a ship, as illustrated in Fig. 2.

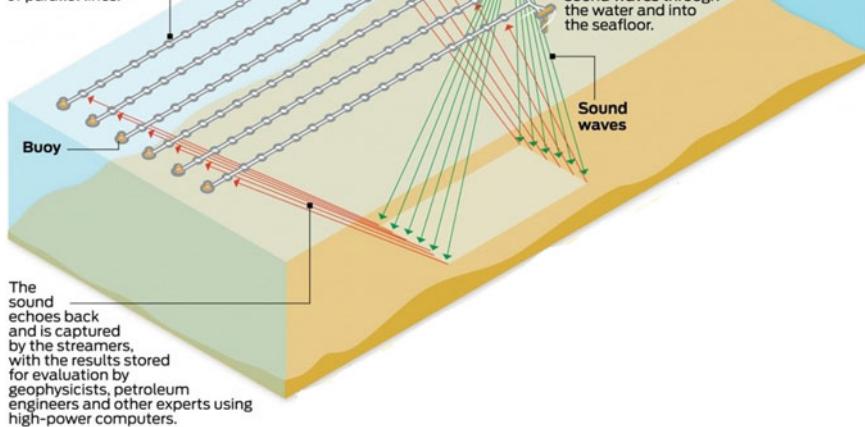
### Seismic changes

The search for oil nearly always involves seismic imaging – determining what lies underground or beneath the seafloor by evaluating echoes from blasts of sound.

In recent years, offshore seismic operators have put a few twists on the process.

#### The basics

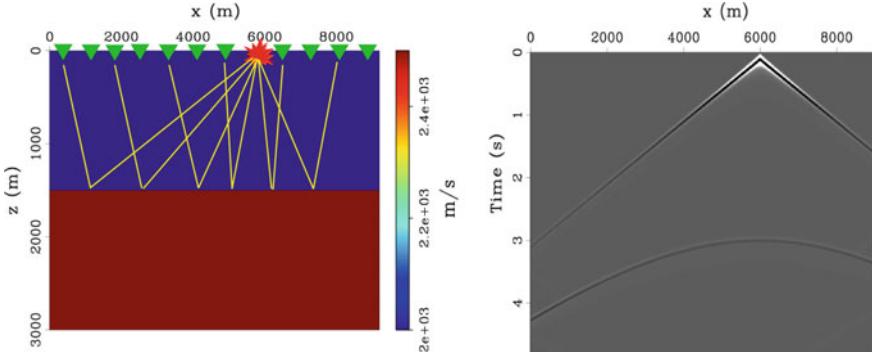
Conventional offshore seismic surveying uses a single vessel towing arrays of sensors called **streamers** in a series of parallel lines.



**Fig. 2** Offshore seismic data acquisition (Source Houston Chronicle, BP, Schlumberger, Fairfield Nodal)

When energy is emitted from the source, it propagates through a highly heterogeneous medium (i.e., subsurface) which in turn creates reflections, refractions and diffraction effects that are recorded at the receiver (sensors) locations. As these recorded events are created due to changes in subsurface rock properties, inherently they contain information about the subsurface from whence they originated. The goal of seismic tomography and seismic imaging in general is to reconstruct the subsurface (earth model) that created the recorded seismic data.

With only one source firing and a finite number of receivers, only a limited portion of the subsurface target of interest can be sampled. Therefore, in order to adequately illuminate the subsurface, it is required that the source and array of receivers be positioned at multiple spatial locations. In reflection seismic terminology, the data obtained from the source firing at a single position  $\mathbf{x}_{s_i}$  into an array of receivers  $\mathbf{x}_{r_i}$ ,  $i = 1, \dots, N_r$  where  $N_r$  is the total number of receivers recording during a source firing is known as a “shot gather”. Modern reflection seismic data acquired for industrial purposes are composed of hundreds of thousands of shot gathers. Figure 3 depicts the ray paths (discrete approximation of a wavefront) associated with a shot gather for a single layer subsurface model and synthetic data recorded as a result of finite-difference modeling with a point source located at the position  $\mathbf{x}_s$ . Note that as the source moves along the surface with a dense array of receivers, subsurface



**Fig. 3** (left) Raypaths for a seismic shot gather acquired over a flat layer earth. (right) Simulated data using finite-difference modeling. The linear event corresponds to the wave that travels directly from the source to the receivers along the surface. The hyperbolic event corresponds to the reflection of the wave off of the layer interface

points will be illuminated multiple times. To take advantage of this data redundancy, seismic data are typically transformed into midpoint and half-offset coordinates via the following relations

$$\mathbf{x}_{m_i} = \frac{\mathbf{x}_{s_i} + \mathbf{x}_{r_i}}{2},$$

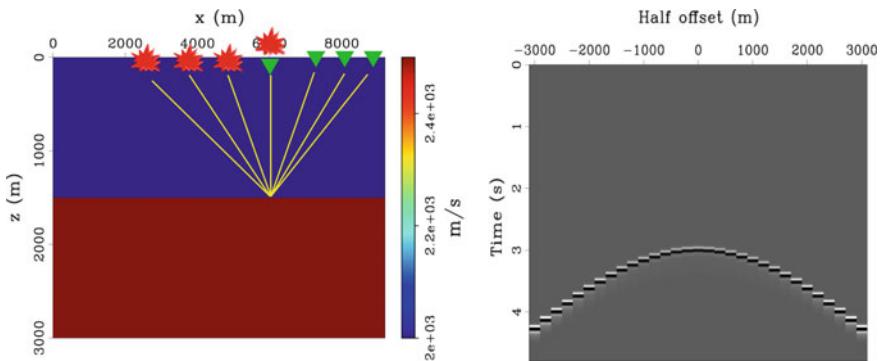
$$\mathbf{x}_{h_i} = \frac{\mathbf{x}_{s_i} - \mathbf{x}_{r_i}}{2}.$$

where  $\mathbf{x}_{m_i}$  and  $\mathbf{x}_{h_i}$  are the midpoint and half-offset coordinates respectively. Figure 4 shows the resulting raypaths and data that arise from sorting the data in Fig. 3 into the midpoint and half-offset domain. As this collection of records is for a fixed midpoint and several offsets, this type of data is known as a common-midpoint gather. The processing of seismic data for velocity model building in general is performed with the data transformed into the midpoint and half-offset coordinates.

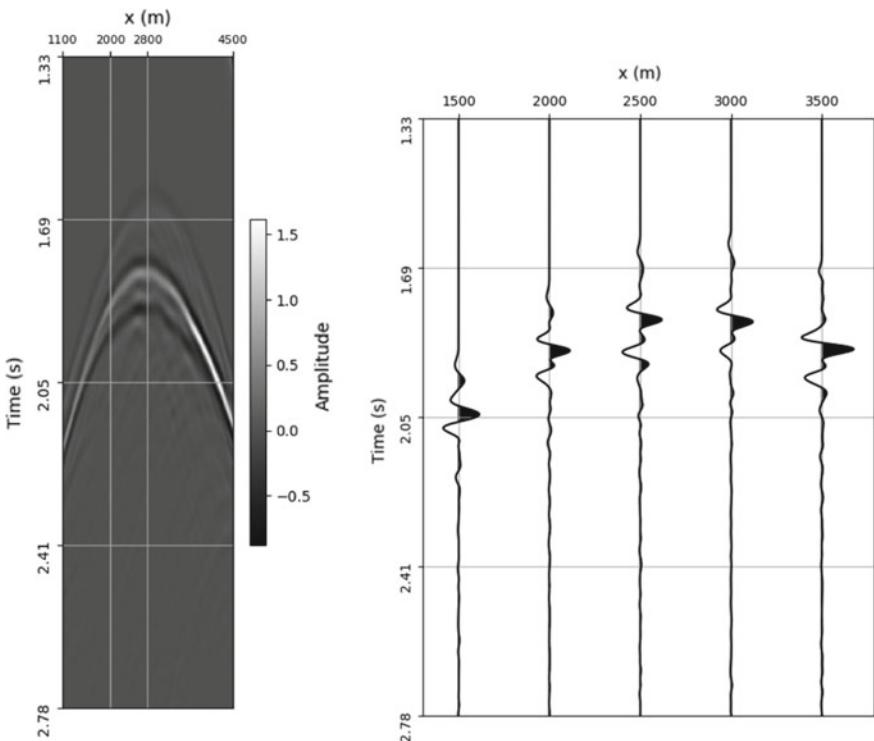
In Fig. 5, a selected group of traces from a more complex subsurface recording is presented. The field recordings—depending on the origin—are like the above depicted ones or more complex, therefore direct interpretation of subsurface structure is ruled out and this originates the need for advanced techniques to transform this data into usable models.

## 2.2 Seismic Tomography

The study of seismic tomography has spanned the past several decades and continues to be part of ongoing research [9]. While there exist many ways to formulate this



**Fig. 4** (left) Raypaths for a seismic common midpoint gather acquired over a flat layer earth. (right) The synthetic data from Fig. 3 sorted into midpoint and half-offset coordinates and with a mute applied to the direct wave



**Fig. 5** (left) Windows in time and space on a shot gather from a complex subsurface model simulated with finite-difference modeling, therefore very high signal-to-noise ratio. (right) Selected traces from the shot gather of the left, traces presented as wiggles, where characteristics of the signals are shown

reconstruction problem [10–12], it can be expressed most generally as the following minimization problem:

$$\mathbf{m}^* = \arg \min_{\mathbf{m}} \{L(f(\mathbf{m}), \mathbf{d})\}, \quad (1)$$

where  $\mathbf{m}$  represents the earth model that we desire to recover,  $\mathbf{d}$  represents the recorded seismic data,  $f(\mathbf{m})$  is a physics-based modeling that generates synthetic data from a prescribed earth model,  $L$  is a loss function that measures the misfit between the recorded data and the simulated data, and  $\mathbf{m}^*$  is the optimal earth model that minimizes the loss  $L$ . While a highly complex  $\mathbf{m}$  that informs us of many different earth properties (elastic moduli, density, viscoelastic parameters, etc.) is generally desired,  $\mathbf{m}$  commonly represents a three-dimensional acoustic wavespeed model. This choice of  $\mathbf{m}$  generally leads to the scalar acoustic wave equation as the choice for our physics-based simulation  $f(\mathbf{m})$ . Further simplifications can be made in taking the high-frequency limit of the scalar acoustic wave equation which results in the eikonal equation [13]. While the wave equation describes the propagation of waves and calculates synthetic seismograms (waveforms), the Eikonal equation is based on ray theory and calculates traveltimes. Regardless of the model parameterization and physics-based forward model used to fit the recorded geophysical data, the relationship between the data and the desired earth model is nonlinear. Therefore, a nonlinear optimization algorithm is required in order to minimize the loss-function (in Eq. (1)). Additionally, because  $f(\mathbf{m})$  is in general very computationally expensive to evaluate, local/gradient-based methods for optimization must be used as opposed to global optimization methods. Using only the gradient information of the loss function can result in convergence to a local minimum and therefore unsatisfactory solutions. Additionally, because for reflection seismic surveys the data are recorded at the earth's surface, the data do not contain all of the necessary information to define a velocity model that varies arbitrarily space. This therefore implies that Eq. (1) defines a non-linear ill-posed optimization problem. In using a deep-learning approach, while we still face this issue of non linearity and ill-posedness, we do not rely on an accurate solution of the wave-equation, but rather directly learn a tomographic operator from many training examples that consist of the seismic data as feature and the velocity model as label.

### 3 Seismic Tomography via Deep Learning

#### 3.1 Deep Neural Networks for Inverse Imaging Problems

Seismic tomography is an inverse imaging problem, in which the observation model can be represented as:

$$\mathbf{d} = f(\mathbf{m}) + \epsilon, \quad (2)$$

where  $\mathbf{d}$  is the observed seismic data,  $\mathbf{m}$  is the unknown earth model,  $f()$  is a mapping operator and  $\epsilon$  is noise. While inverse imaging problems can be solved using analytic models, recent works [14–16] (and references within), argue that state-of-the-art results for a variety of inverse imaging problems can be obtained using deep learning methods. Following this line of work, we have proposed a novel approach [7] that implements the tomography operator using a convolutional neural network (CNN), whose coefficients are learned in a data-driven approach [17]. The tomography process is depicted in Fig. 6, and it performs reconstruction of the velocity model from raw seismic traces, or from features computed from raw seismic traces. In a real-life application, the ground-truth model is unavailable, and the tomography operator is designed to minimize the difference between the reconstructed velocity model and the (unavailable) ground-truth one. The input to the tomography operator  $\mathbf{T}(\mathbf{d}; \theta)$  is a set of seismic traces (or their features)  $\mathbf{d}$ , and it is parameterized by a coefficients vector  $\theta$ . The tomography operator approximates the inverse mapping operator  $f^{-1}()$ , and its output is the predicted velocity model  $\hat{\mathbf{m}}$ . In the statistical learning framework, the tomography operator is learned using a collection of  $N$  training example pairs  $\{\mathbf{d}_i, \mathbf{m}_i\}_{i=1}^N$ , where the data  $\mathbf{d}_i$  denotes the set of seismic traces (i.e. seismic gather) or their features, as generated by wave propagation simulation using the  $i$ -th velocity model  $\mathbf{m}_i$  (the  $i$ -th label). The average misfit between the ground truth models and their predicted versions, also known as the empirical risk, is defined by:

$$\mathbf{J}(\theta) = \frac{1}{N} \sum_{i=1}^N L(\mathbf{m}_i, \hat{\mathbf{m}}_i), \quad (3)$$

where  $L(\mathbf{m}_i, \hat{\mathbf{m}}_i)$  is the loss function that measures the misfit between the ground truth velocity model and its prediction  $\hat{\mathbf{m}}_i = \mathbf{T}(\mathbf{d}_i; \theta)$ . The tomography operator is learned by minimizing the empirical risk:

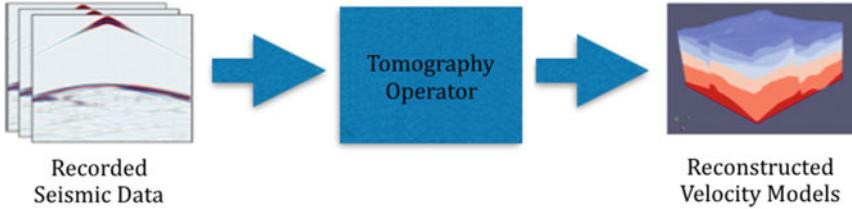
$$\hat{\theta} = \arg \min_{\theta} \mathbf{J}(\theta). \quad (4)$$

The loss function employed in this work is the squared  $L_2$ -norm of the pixel-wise difference  $\hat{\mathbf{m}} - \mathbf{m}$ , given by:  $L(\mathbf{m}_i, \hat{\mathbf{m}}_i) = \|\mathbf{m}_i - \hat{\mathbf{m}}_i\|_2^2$ , which is frequently used in regression problems, and leads to the following risk minimization problem:

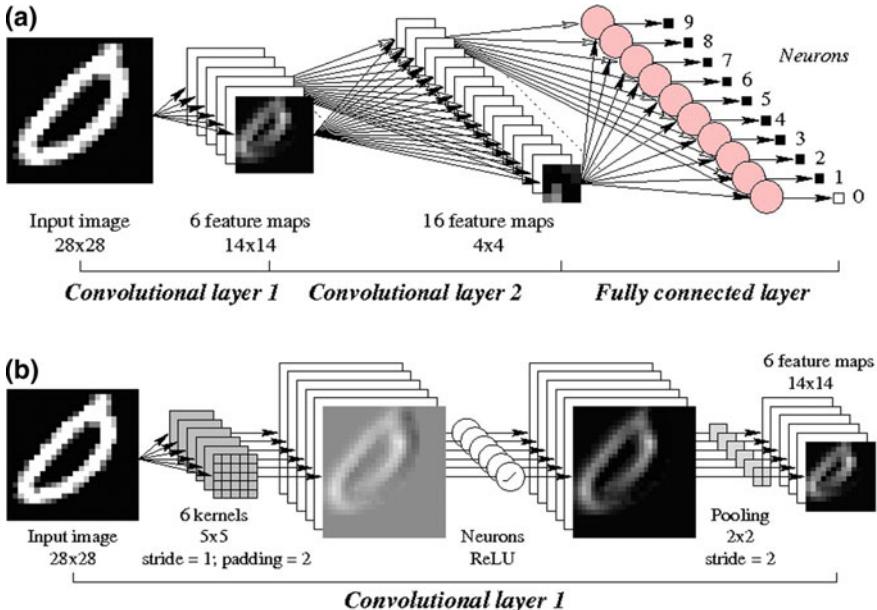
$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N \|\mathbf{m}_i - \mathbf{T}(\mathbf{d}_i; \theta)\|_2^2. \quad (5)$$

In addition, regularization [18] of network parameters is optionally applied by an additional term  $\mathbf{R}(\theta)$ , leading to the following minimization problem:

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N \|\mathbf{m}_i - \mathbf{T}(\mathbf{d}_i; \theta)\|_2^2 + \lambda \mathbf{R}(\theta), \quad (6)$$



**Fig. 6** Tomography reconstruction of velocity models from recorded seismic data



**Fig. 7** Convolutional Neural Network (CNN): **a** a CNN with two convolutional layers and one fully-connected layer; and **b** zoom into the first convolutional layer (Source [20])

where  $\lambda \geq 0$  controls the weight of the regularization term  $\mathbf{R}(\theta)$ , which is often defined as Ridge regression  $\mathbf{R}(\theta) = \|\theta\|_2^2$  or Lasso regression  $\mathbf{R}(\theta) = \|\theta\|_1$ .

The tomography operator is implemented by a CNN, and thus can be represented as a hierarchical composition of  $k$  non-linear functions, each representing one of the  $k$  layers of the network:

$$\mathbf{T}(\mathbf{d}; \theta) = g_k(g_{k-1}(\cdots g_2(g_1(\mathbf{d}; \theta_1); \theta_2); \theta_3); \theta_k), \quad (7)$$

where  $\theta = [\theta_1, \theta_2, \dots, \theta_{k-1}, \theta_k]^T$ , and each function represents either a fully-connected (FC) or a convolutional layer [17, 19], as illustrated in Fig. 7.

### 3.2 Velocity Semblance as Input Feature for Deep Networks

Feature extraction is an optional step in our workflow as it can accelerate the training of the CNN by providing it with the most relevant data for learning. Our ML platform is capable of handling diverse network architectures and data, but given the focus on learning a tomographic operator from the data, we perform what is known as velocity (main subsurface model attribute) analysis and use its output as the input feature space.

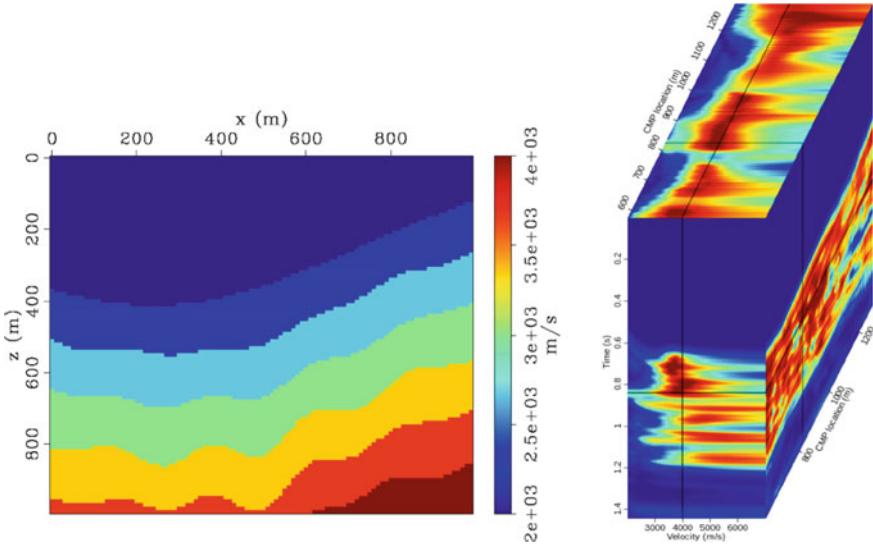
To perform velocity analysis, we first transform the data into the midpoint half-offset coordinates as discussed previously. Then, we perform a time shift to each offset  $h$  of the common-midpoint gather in order to flatten the reflection (which has a hyperbolic shape) along the offset direction. This time-shift is a function of the half-offset  $h$  and the velocity in the medium  $V$  and can be calculated via the following relationship

$$t^2(h, V) = t_0^2 + \frac{h^2}{V^2}, \quad (8)$$

where  $t$  is the travel time of the hyperbolic event and  $t_0$  is the time at which the data were recorded. Note that Eq. 8 describes the shape of a hyperbola which is exactly the shape of the recorded reflection shown in Fig. 4. Performing this time shift requires that the medium velocity be known a priori (which in the case of VMB is not). Therefore, trial velocities are prescribed in order to flatten the reflection event and then the following coherency measure is used in order to measure the flatness of the time-shifted event

$$s[i] = \frac{\sum_{j=i-M}^{i+M} \left( \sum_{k=0}^{N-1} q[j, k] \right)^2}{N \sum_{j=i-M}^{i+M} \sum_{k=0}^{N-1} q[j, k]^2}, \quad (9)$$

where  $q[j, k]$  is the time-shifted common-midpoint gather for a particular velocity  $V$  and  $j$  and  $k$  are the time and offset indices respectively. The inner sum over all  $N$  offsets sums the time-shifted event along the offset direction. Therefore, the flatter the event (or the closer the prescribed velocity is to the true velocity), the greater the output of the sum. The outer sum is an average in time over a window of  $2M + 1$  samples. The output  $s[i]$  coherency measure is known as semblance [8] and is often the first step towards building a velocity model in reflection seismology. Performing this calculation for multiple midpoints, a semblance cube which has axes of time, velocity and midpoint can be created. The right half of Fig. 8 shows an example of a semblance cube for the velocity model shown in the left half of Fig. 8. Note that while the semblance cube does not offer very high resolution information about the velocity model. Rather, it gives an overall trend of how the velocity increases with depth from midpoint to midpoint.



**Fig. 8** (left) 2D Synthetic earth model, layers of sediments in a simple depositional system, velocity ranges between 2000 and 4500 m/s. Horizontal coordinate is and vertical represents depth. (right) Example of a calculated semblance cube for the model in left. In our case, during training, models like left are the labels and semblance cubes the input data

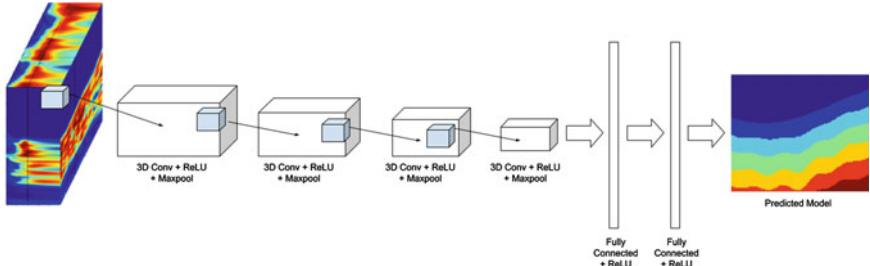
## 4 Semblance-Based CNN Results

In this section we first describe the experimental setup, including network architecture, datasets, hardware and software and metrics used for quantitative analysis. Second, we present the qualitative and quantitative results and discussion.

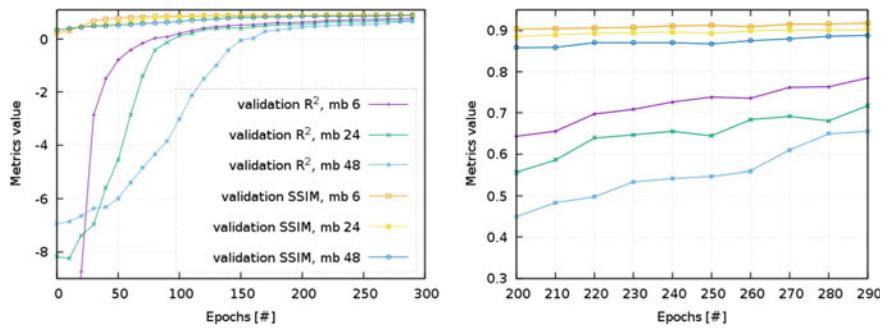
### 4.1 Experimental Setup

The network architecture is composed of four 3D convolutional layers (64 filters with kernel size of  $6 \times 6 \times 6$ ) and two fully connected layers. Each layer employs a ReLU activation function. In addition, max-pooling, batch normalization and dropout with probability of 0.25 are deployed after each convolutional layer, as depicted in Fig. 9. The loss function is mean squared error and Nesterov ADAM [21, 22] optimizer is used. The network is implemented in python using TensorFlow [23] and Keras [24] as DL supporting frameworks.

The training reaches early stopping on around 250 epochs, in about 6 hours running on one high performance computing (HPC) node sporting four general purpose graphical processing units (GPGPUs) NVIDIA K80 [25] in data parallelism fashion. In this parallel execution mode, the model is copied to all computing units and



**Fig. 9** Semblance-based 3D CNN architecture: the semblance cube is the input feature to the network, which includes four 3D convolutional layers and two fully connected layers. Each convolutional layer is composed of 3D kernels, ReLU activation per kernel, Maxpool, Batch Normalization and Dropout layer



**Fig. 10** (left) The plot shows the metrics value across training time. The vertical axis represent the metric value and the horizontal axis represents time in epoch units, where one epoch is a complete sweep through the training dataset. (right) A detailed view of the left plot around an area of interest. Plots share color codes and mb stands for minibatch

then the training data are evenly split and distributed among the computing units to be solved. Inference per model is a matter of seconds, which is extremely appealing when large amount of data is predicted or multiple velocity scenarios are under investigation.

Two datasets are prepared for both training and testing our model. In the first dataset, the velocity models only contain layers with velocities that increase with depth. Additionally, the layers exhibit both undulation and dip (Fig. 11). The second dataset consists of similar velocity models as the first dataset, but now a portion have been augmented with salt bodies. To add a factor of realism to these models, the shape of the salt bodies were extracted from earth models that were the end result of real life exploration projects in the Gulf of Mexico. Moreover, this dataset also contains velocity models without salt bodies. Each dataset consists of 6400 semblance cubes and the corresponding velocity model labels of size  $100 \times 100$  grid points (the size of the output layer). For validation and testing purposes, we separated 1600 data/label pairings.

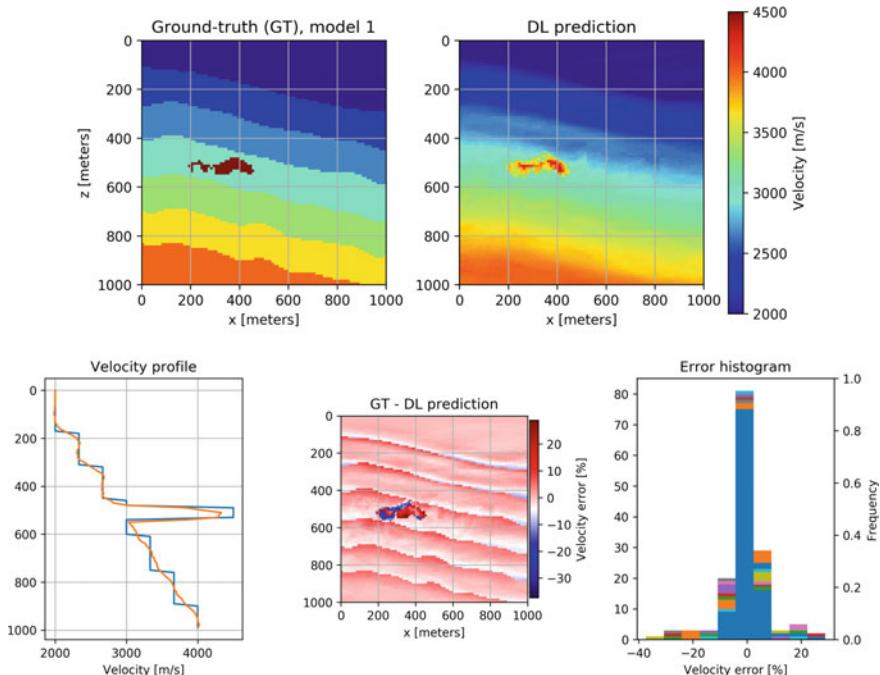
## 4.2 Quantitative Metrics

In terms of quantitative metric for model quality comparison, we decided to recourse to the widely accepted standard metric in image-dominated fields, the structural similitude index metric (SSIM) [26] and peak signal-to-noise ratio (PSNR). SSIM differs from traditional objective metric since it is based on structural degradation rather than error or general distortion of the images.

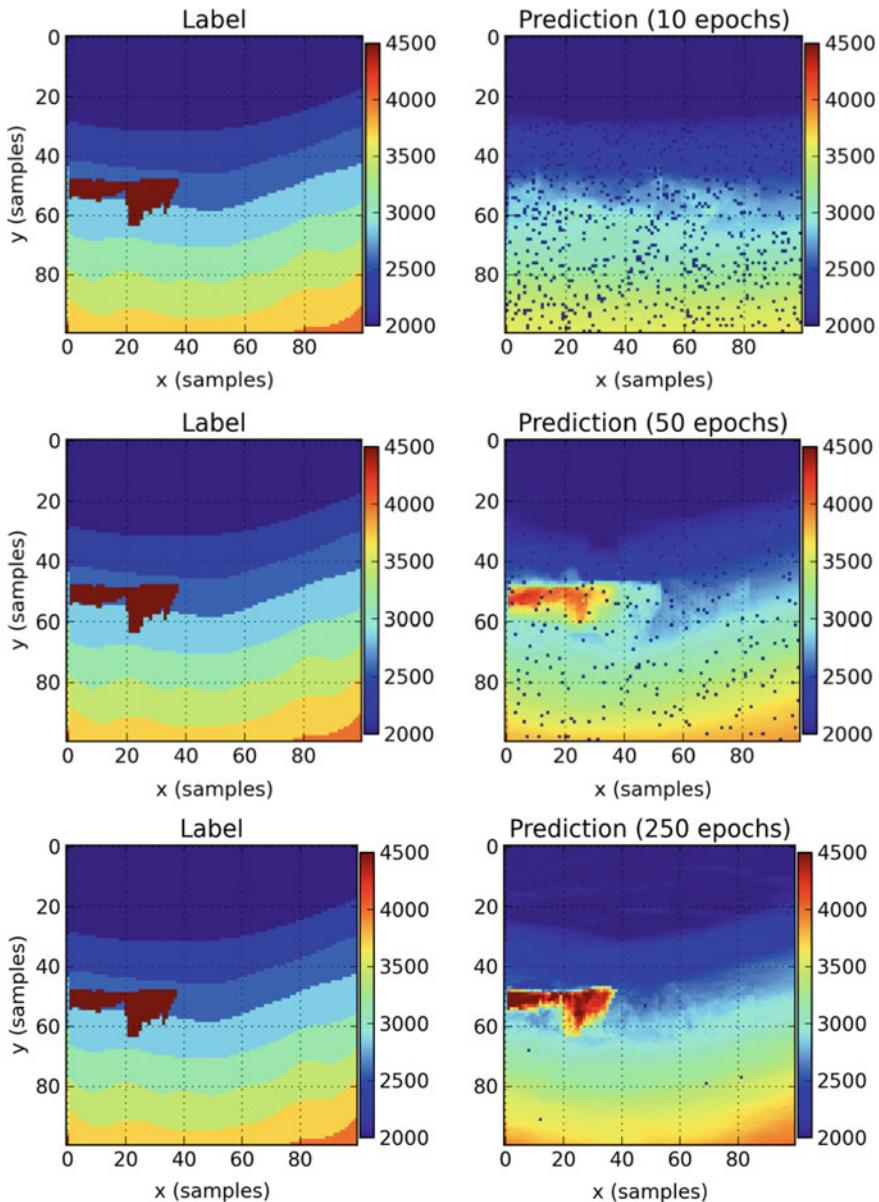
From statistical perspective the robustness of the model is measure with  $R^2$  score (coefficient of determination).

## 4.3 Results and Analysis

The prediction accuracy metrics on the testing set for the first dataset (earth models only containing layers) are 0.812 for the  $R^2$  score and 0.919 for the SSIM.  $R^2$  score for



**Fig. 11** (top, left) model 1 of the test dataset, includes salt bodies, which have high velocity and tend to distort classical modeling. Salt bodies are key in offshore hydrocarbon exploration. (top, right) prediction, where vertical axis represents depth and horizontal axis represents lateral offset. (bottom, left) comparison of the velocity profile for  $x = 400$ , where the vertical axis represents depth in meters. (bottom, center) absolute error between the ground-truth and prediction and (bottom, right) the corresponding error distribution



**Fig. 12** Improving model reconstruction (for one model in the testing dataset) as the learning process sweeps through the training dataset

the test set with the second dataset is 0.741 and the SSIM is 0.892. The convergence of these metrics with respect to epoch can be observed in Fig. 10. The convergence curves shows the influence of different batch sizes on the performance for both metrics, although the effect is most noticeable on the  $R^2$  metric, which converges later than the SSIM metric. As expected, the task of predicting a model with salt bodies is more difficult and therefore the performance is lower for dataset two than dataset one. It is difficult to learn the size, shape, and location of these salt bodies from the input data space. Furthermore, the datasets are relatively small for the task at hand. The main impediment to obtaining more training data is the computationally expensive step of generating features via finite difference wave propagation and calculating the semblance feature.

Qualitatively, the overall performance trend is positive, the salt bodies are mostly located properly and the surrounding formation resembles the labels in structure and velocity value (see Fig. 11), thus making the predicting model valid.

The main structural elements of the predicted model matches the ground-truth. The predicted expression and location of the salt body in Fig. 11 is remarkable. In particular, the velocity profile shows that the velocity trend is perfectly recovered, only missing the sharp interfaces between layers. In Fig. 12 we observe how a model from the validation set is learned as the training of the network progresses (by epochs). The first prediction (Fig. 12, top row) shows a model with low velocity in a gradient-based background, with many unresolved samples (blue dots). After few epochs (Fig. 12, center row) the predicted model corrects the deeper sections towards higher velocity and the salt body is clearly reconstructed. Finally, the model prediction is complete (Fig. 12) and even fine grained details of the salt body are satisfactory resolved.

## 5 Industry Baseline: Full Waveform Inversion

### 5.1 *Industry VMB Methods*

Many seismological techniques exist to estimate material properties in the shallow mantle of the earth. Ray tracing methods rely on high frequency ray theory approximation and picked arrival times of body waves to invert for optimal shear and pressure velocity models [27, 28]. These techniques are restricted to smooth medium predictions and fall short when surface wave amplitudes dominate body wave arrivals [29]. Full-waveform inversion (FWI) attempts to achieve this by iteratively simulating the seismic experiment and updating the earth model until the simulated seismic data matches the recorded seismic data in a least squares sense [30]. By fully modeling how energy propagates through the subsurface, FWI is more likely than other methods to find accurate representations of the earth’s material properties [31].

## 5.2 Full Waveform Inversion

FWI uses the entire seismic wavefield recording, that being all recorded frequencies and locations, to invert for earth model parameters beneath the surface. The goal of FWI is to find some earth model that minimizes the distance between modeled seismic data, which is a function of the earth model, and recorded seismic data, which was gathered in the field. When we have changed the earth model in such a way that the modeled data very closely resembles the recorded data, we assume we have found an earth model that is representative of the true earth model. Albert Tarantola [30] was the first to propose solving for earth parameters with such an inverse solution. In exploration geophysics, FWI is a topic of intense study and is at the forefront of earth model building from seismic data [32]. That being said, it is plagued with numerous limitations including high computational cost, extreme sensitivity to the choice of starting model, and unwanted convergence to incorrect earth model solutions. Moreover, when these limitations are properly accounted for and addressed, FWI is regarded as an area of development that may rectify the gap between low and high wavenumber earth model building and represent an all-inclusive solution to seismic exploration. For this reason we have chosen it as a baseline method to compare the velocity model prediction results of the ML approach defined previously. If ML can compete with the current cutting edge industry techniques, it will surely make waves in the exploration community.

More verbosely, consider the  $i^{th}$  shot of a seismic survey  $\mathbf{d}_i^{obs}$  where  $i = 1, 2, \dots, M$ . Further, consider some modeled data,  $\mathbf{d}_i^{mod}$ , which is the synthetic recreation of the  $i^{th}$  observed experiment. We can define the distance between the observed and modeled data as the  $L_2$  norm of the two vectors,

$$L_2(\mathbf{d}_i^{mod}, \mathbf{d}_i^{obs}) = \|\mathbf{d}_i^{mod} - \mathbf{d}_i^{obs}\|_2. \quad (10)$$

To create the modeled data we use some wave equation operator,  $f_i$ , which represents a single seismic experiment.  $f_i$  is a function of the earth model,  $\mathbf{m}$ , and maps from the earth model space into the data space,  $f_i(\mathbf{m}) = \mathbf{d}_i^{mod}$ . In our case  $\mathbf{m}$  represents  $1/v^2$ , the inverse of the squared pressure wave velocity, at each point in the subsurface. Many wave equation formulations can model how seismic energy propagates through the earth. Generally speaking, the more complex and accurate the wave equation, the more computationally expensive the wave modeling becomes. For our purposes we use the acoustic, constant density, isotropic wave equation [33].

$$(\mathbf{A} - \mathbf{MD}_2) \mathbf{p} = \mathbf{f}, \quad (11)$$

where  $\mathbf{A}$  is a laplacian operator,  $\mathbf{D}_2$  is a second time derivative operator,  $\mathbf{M}$  is composed of  $nt$  copies of the flattened  $\mathbf{m}$ , where  $nt$  is the total number of time samples in the seismic recording,  $\mathbf{p}$  is the pressure wavefield, and  $\mathbf{f}$  is the injected seismic source. This wave equation assumes the earth can be represented by a single elastic parameter, pressure wave velocity, is isotropic, has constant density, and has a zero

shear modulus. We can represent this wave equation operator with a matrix,  $H(\mathbf{m})$ , and solve for the wavefield  $\mathbf{p}_i$ :

$$H(\mathbf{m})\mathbf{p}_i = \mathbf{f}_i \quad (12)$$

$$\mathbf{p}_i = H^{-1}(\mathbf{m})\mathbf{f}_i, \quad (13)$$

where  $\mathbf{p}_i$  represents the wavefield resulting from the  $i$ th seismic experiment in the entire domain. We can use an operator,  $K$ , to extract the wavefield at the point receiver locations to arrive at the modeled data,  $\mathbf{d}_i^{mod}$ :

$$\mathbf{d}_i^{mod} = K\mathbf{p}_i = KH^{-1}(\mathbf{m})\mathbf{f}_i = f_i(\mathbf{m}). \quad (14)$$

Using this wave equation operator we can define a scalar function,  $J(\mathbf{m})$ , which sums the  $L_2$  difference between modeled and observed seismic data over all experiments:

$$J(\mathbf{m}) = \sum_{i=1}^N ||f_i(\mathbf{m}) - \mathbf{d}_i^{obs}||_2^2. \quad (15)$$

Here we have arrived at what is referred to as the FWI objective function. The model that reaches the minimum of this objective function is the solution to the FWI problem and the model that is our best estimate of the velocity profile beneath the surface.

Solving this inverse problem, that is finding the model that minimizes  $J(\mathbf{m})$ , is notoriously difficult for a variety of reasons. Primarily, the objective function is nonlinear with respect to  $\mathbf{m}$ , which means a perturbation in the earth model is not linearly mapped into the modeled data. It follows that the numerous, well studied strategies to solve linear least squares inverse problems are useless to us. Instead we must resort to nonlinear regression techniques for which there is no general theory for finding the optimal model parameters [34]. Iterative methods are a popular choice for solving nonlinear inverse problems and rely on the gradient of the objective function at the current model iteration,  $\mathbf{m}_j$ , to update the model parameters to find the next model iteration,  $\mathbf{m}_{j+1}$ .

$$\mathbf{m}_{j+1} = \mathbf{m}_j + \alpha_j \mathbf{s}_j. \quad (16)$$

The next model,  $\mathbf{m}_{j+1}$ , is found by summing the current model,  $\mathbf{m}_j$ , to some search direction,  $\mathbf{s}_j$ , scaled by a step length,  $\alpha_j$ . There are many ways to compute the search direction,  $\mathbf{s}_j$ . We use the nonlinear conjugate gradient method in which:

$$\mathbf{s}_j = \mathbf{s}_{j-1} + \beta \nabla J(\mathbf{m}_j), \quad (17)$$

where  $\mathbf{s}_{j-1}$  is the previous search direction and  $\beta$  is the conjugate direction coefficient, and  $\nabla J(\mathbf{m}_j)$  is the gradient of the objective function at the current model.

Furthermore,  $B(m_j)^*$  is the adjoint of the wave equation operator linearized around the current model iteration applied to the difference between the modeled and observed data:

$$\nabla J(\mathbf{m}_j) = - \sum_{i=1}^N B(\mathbf{m}_j)^*(f_i(\mathbf{m}) - \mathbf{d}_i^{obs}) \quad (18)$$

To put it concisely, at each iteration of FWI we use the gradient of the objective function to update the earth model in order to reduce the value of the objective function. We stop iterating when the objective function reaches zero or, more realistically, once it stops reducing.

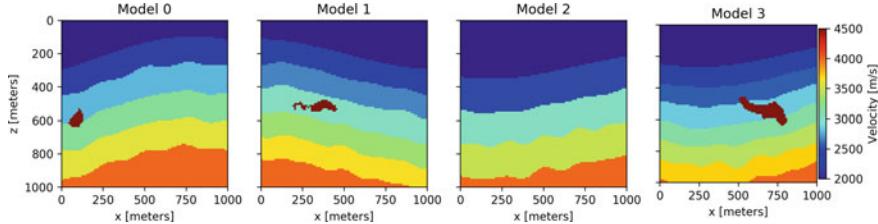
However, the nonlinearity of the FWI objective function means it is not convex with respect to the earth model. Gradient descent methods, like the one described above, will fall into a local minimum, that is find an earth model at which the objective function stops reducing but does not represent the global minimum of the objective function. In order to avoid local minima, the initial model used in the inversion scheme,  $\mathbf{m}_0$  must be fairly close to the true model. Herein lies one of the largest restrictions of FWI, that being we must start from an earth model that is fairly close to the true model in order for the gradient based optimization algorithm to converge to the true solution.

Many methods exist and extensive research continues to find ways to avoid these convergence issues. A highly effective and widely accepted method is that of [35] which is referred to as multiscale FWI. This technique decomposes the FWI problem by scale and performs conventional FWI with progressively higher bandpasses of the source wavelet and observed data.

### 5.3 Baseline Comparison Setup

To compare the velocity model predictions of the proposed CNN-based approach and FWI, four synthetic seismic surveys are created and used as inputs. The intent is to keep the input data consistent in order to create a fair comparison between each method as in [36]. Below describes the data generation, the parameters of the three experiments, and the quantitative methods used to compare model results.

The synthetic models used to generate the seismic data are 1.8 km in the x direction and 1.4 km in the z direction with grid cell discretization of 10 m. The models parameter used is pressure wave velocity that increases with depth and contains salt bodies of varying shape and size. The velocities range from 2.0 to 4.5 km/s. All four of the models have layer cake backgrounds which are representative of the upper crust of the earth. Three of the four compared models contain high velocity zones, around 4500 m/s, which are characteristic of larger salt diapers that often trap migrating oil and gas. It follows that the industry places large interest on finding and resolving these salt bodies.



**Fig. 13** Ground-truth velocity models

The data itself is generated from 19 shots at the surface with 40 m spacing in the x direction beginning at 520 m. The shot wavelet is a 15 Hz peak Ricker. 144 receivers located at the surface record pressure data. They begin at 180 m in x with 10 m spacing. The wave propagation modeling assumes an acoustic, constant density earth and uses second order approximation in time and eight order in space. Figure 13 illustrates the four models used to compare each method. Note, the data was generated on  $1.8 \times 1.4$  km model but the velocity predictions were made on a  $1.0 \times 1.0$  km subset of the original models.

## 5.4 Experiments

The first experiment is conducted with conventional FWI; 1000 iterations of non-linear conjugate gradient are performed using all frequencies of all modeled shots. The starting model was a linear velocity gradient from 2.0 to 4.5 km/s. A variation of this experiment is also conducted in which 200 conjugate gradient iterations are performed using the predicted model from the CNN as the starting model for FWI. The second experiment is multiscale FWI which performed 150 conjugate gradient inversions over 5 bandpasses of the all modeled shots. The first 4 bandpasses of the data were smoothly tapered at 4, 8, 16, and 32 Hz. The fifth inversion used all frequencies. The starting model for the 4 Hz inversion was a linear velocity gradient from 2.0 to 4.5 km/s. Each progressively higher bandpass inversion uses the final model from the previous bandpass inversion. The third experiment results are obtained by exposing the trained neural network to unseen data, in our case, to unseen semblance cubes from velocity models created by our pseudo-random velocity model generator.

## 5.5 Results

We perform the comparative analysis on four seismic datasets generated from the velocity models in Fig. 13. The comparison is limited to four datasets because of the high computational cost of FWI. In fact, retrieving one multiscale FWI result

takes more time than training the CNN used for the ML approach. After the upfront cost of creating the trained CNN, a single model prediction can be made almost instantaneously. This speaks to the computational cost of ML compared to FWI.

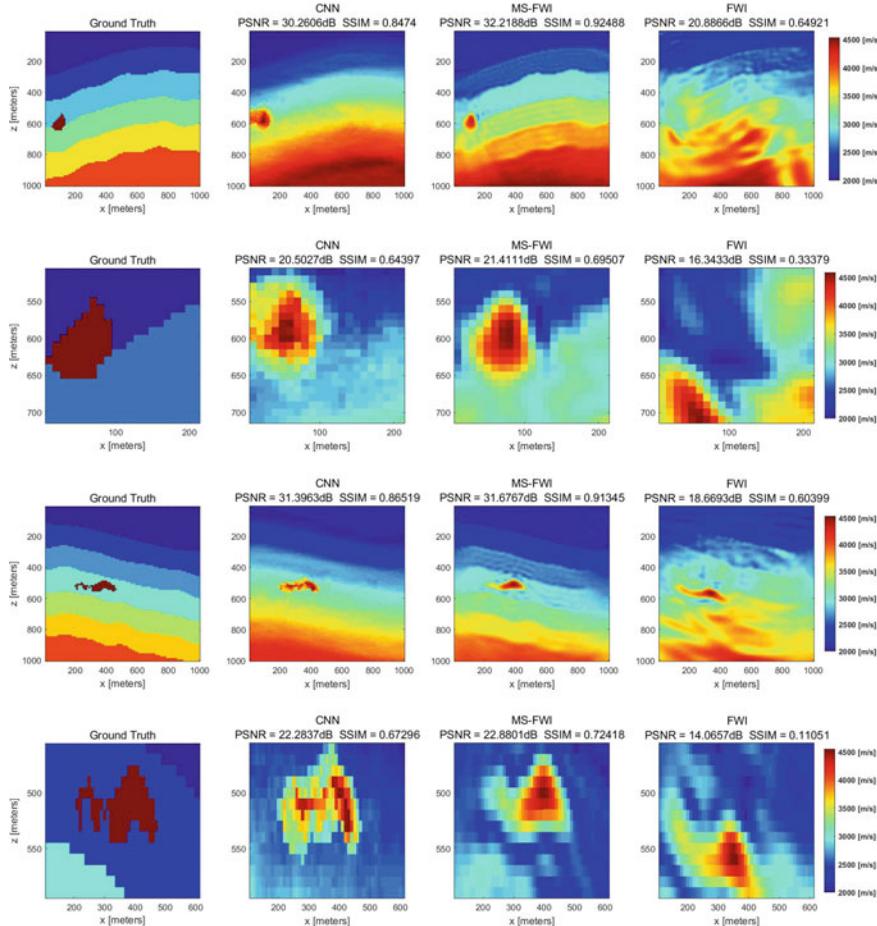
Figures 14, 15, and 16 depict the results of the three VMB methods on the four models both visually and numerically. In Figs. 14 and 15, rows correspond to various models and columns to VMB methods. Since salt diapers are of large interest in the oil and gas community, comparisons are also made over windowed portions of the earth models that contain such bodies. Figure 16 gives a more in-depth look into the results on model 0 by computing difference plots between the true models and each of the VMB method results. This gives intuition on where each method is over or under-performing relative to the others. It also shows the error histograms to illustrate the distribution of velocity errors.

## 5.6 Discussion

A large impact would be made in the exploration seismic community if a method emerged that could construct earth models more effectively than FWI. We claim to have found such a method that leverages ML to show promising result on synthetic experiments. Our approach succeeds where FWI fails, in that ML is more robust, void of human bias, and computationally cheaper.

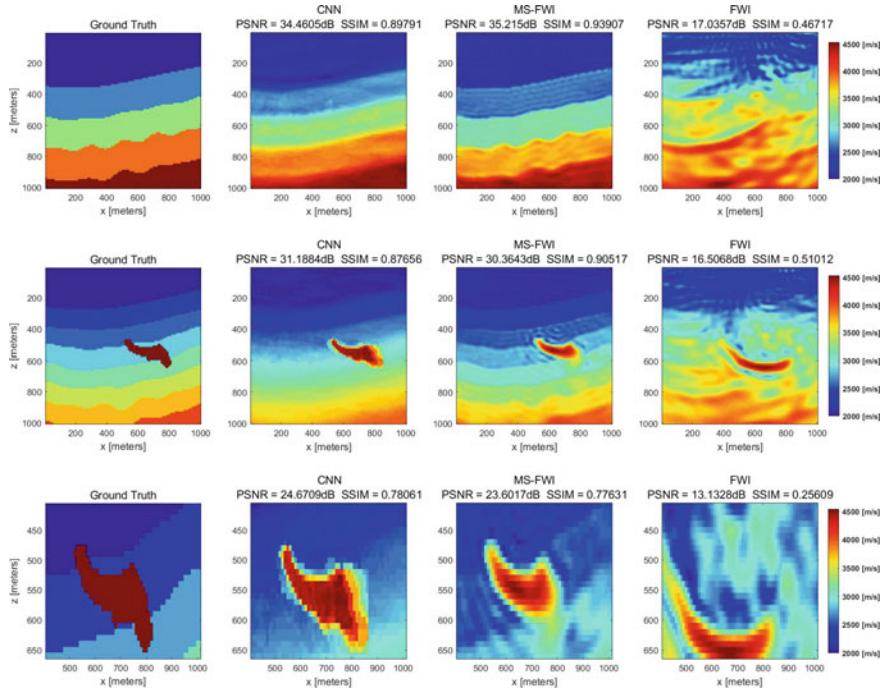
To backup this claim, we must analyze the experimental results visually and numerically in Figs. 14, 15, and 16. When comparing the results of the three approaches, we observe that both the CNN prediction and multiscale FWI were able to recover the original velocity model with good accuracy while the conventional FWI approach fell into a local minimum and was not able to recover a reasonable velocity solution. For example, examine the full view and the zoom view results of model 3, plotted in rows two and three of Fig. 15. The CNN and multiscale FWI methods both resolve the correct salt body location while conventional FWI completely misjudges the depth and shape of the body. Furthermore, the CNN better predicts the complex outline of the salt, including the bottom side which is notoriously difficult in real world applications. In general we find that the output of the DNN is smoother than the velocity estimated via multiscale FWI. This is highlighted in the difference plots of Fig. 16. One can see that the multiscale FWI approach performs better at resolving the interfaces between the layers. This likely due to the fact that when calculating the input semblance cubes, a smoothing occurs which limits the maximum frequency in the semblance cube. Multiscale FWI, however attempts to match modeled and to predicted data that may have a broader range of frequencies. Figures 14 and 15 do show that, generally, multiscale FWI, according to the SSIM metric, outperforms our ML results. But we find visually that the resolution of the salt bodies from the ML approach is more impressive and appealing from an oil and gas interpretation point of view.

Even though the results of DL are preliminary, they are already competitive with FWI. Consider the biases present in each approach. In order for FWI to succeed,



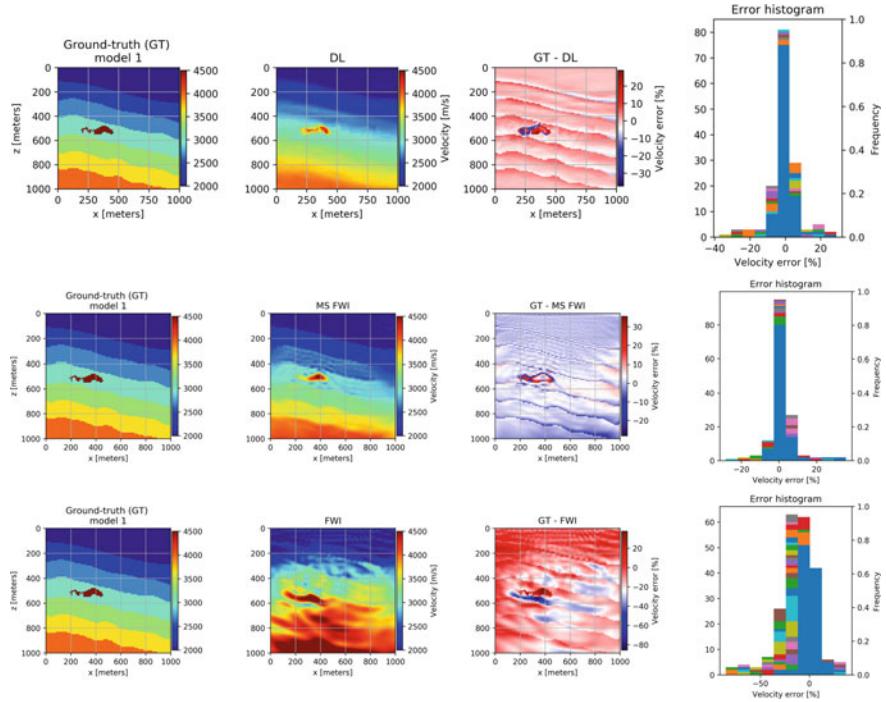
**Fig. 14** Velocity Model Building results comparison: (1st row) Models 0, (2nd row) zoom into the salt body in Model 0, (3rd row) Models 1; and (4th row) zoom into the salt body in Model 1

we needed to use the multiscale scheme. There are dozens of other regularization methods that may or may not work depending on the specific experiment. It is left up to the geophysicist to decide. Furthermore, the sensitivity to the starting model means a priori information on the structure of the earth must be known. In real world scenarios, the starting model used in these experiments, the linearly increasing velocity profile with depth, will not suffice. A fairly detailed starting model must be constructed by the geophysicist beforehand. Alternatively, the ML approach did not need any handpicked regularization of the input data and it requires no starting model. ML retrieved competitive results without any human bias. Furthermore, FWI has been in development for 20 years and our ML method is also in its infancy. If we can recover sharper velocity model results with ML, and thus beat the FWI



**Fig. 15** Velocity Model Building results comparison: (1st row) Models 2; and (2nd row) Models 3, (3rd row) zoom into the salt body in model 0

results, there is nothing stopping ML from replacing FWI. Beyond comparing the velocity model results, we must address an equally important aspect, computational efficiency. The ML and FWI results were computed at different high performance computer cluster facilities, making a direct computational comparison difficult. But, we will find that examining precise clock cycles is not necessary because, by rough estimation, ML is orders of magnitude more efficient. Consider that to perform 1000 iterations of nonlinear conjugate gradient to recover the multiscale FWI results took about two days on a busy Stanford University computer cluster. Now, of course, the modeling and inversion codes used were for academic purposes and were therefore not fully optimized. But, the earth models used are also fairly small,  $1\text{ km} \times 1\text{ km}$ , by industry standards. If more efficient code was used on larger models, the compute time would most likely remain on the same order of magnitude; days. Now, consider that training the CNN model to map from semblance cubes to velocity models took about a day to finish. If larger models are used this may increase. Regardless, one may conclude that both methods are about equally efficient as both take on the order of days to finish. But, herein lies a critical difference between the two approaches; the CNN model is *reusable*. Once the training is finished, mapping from a single new dataset to a velocity model is nearly instant. Whereas, mapping a new dataset using multiscale FWI would take days to finish. The cost of the ML approach is all



**Fig. 16** Comparison of tomography results from the DL and FWI for model 0. Leftmost column shows ground-truth (label), second from left shows the prediction from the DL (top), the multiscale (MS) FWI result (middle) and the standard FWI result (bottom). Third column from left shows the difference between the ground truth and the prediction as a percentage of the velocity error. The last column shows the percentage of velocity errors for each sample binned and plotted in a histogram form. When comparing the prediction of the DNN to the MS FWI result, we observe that the DNN has difficulty in resolving sharp interfaces. Also note that a MS FWI approach was necessary to avoid cycle skipping that is apparent with the conventional FWI result

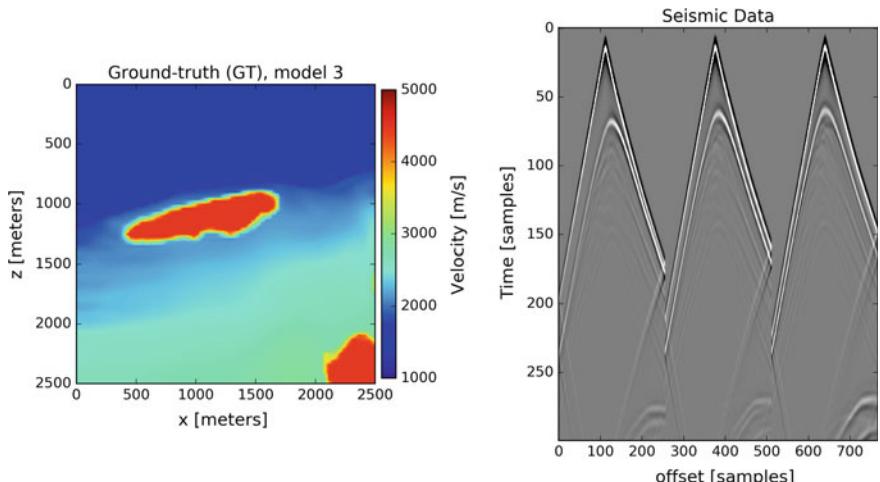
upfront and can be reused an infinite amount of times to make instant predictions. Nothing about FWI is reused and each application is equally expensive.

We show a new way of doing tomography with ML that leaves human biases and reoccurring high computational cost behind. While the ML results are competitive, they are still beat by a regularized FWI method. But, our ML method is also in its infancy and FWI has been in development for over 20 years. Further progress may reap a ML method that can outperform FWI on all fronts, including model quality. A synergistic approach that utilizes both techniques is also an interesting, and a more realistic proposition. Using the unbiased results of ML as a starting model, FWI could fill in the remaining, sharp contrasts with fewer required iterations. This could quickly produce high quality models completely void of human bias. The broader case we make here is for the revolution of workflows in industry exploration. We see potential for many intermediate steps to be absorbed by ML-driven approaches, and seismic tomography is a stepping stone towards that.

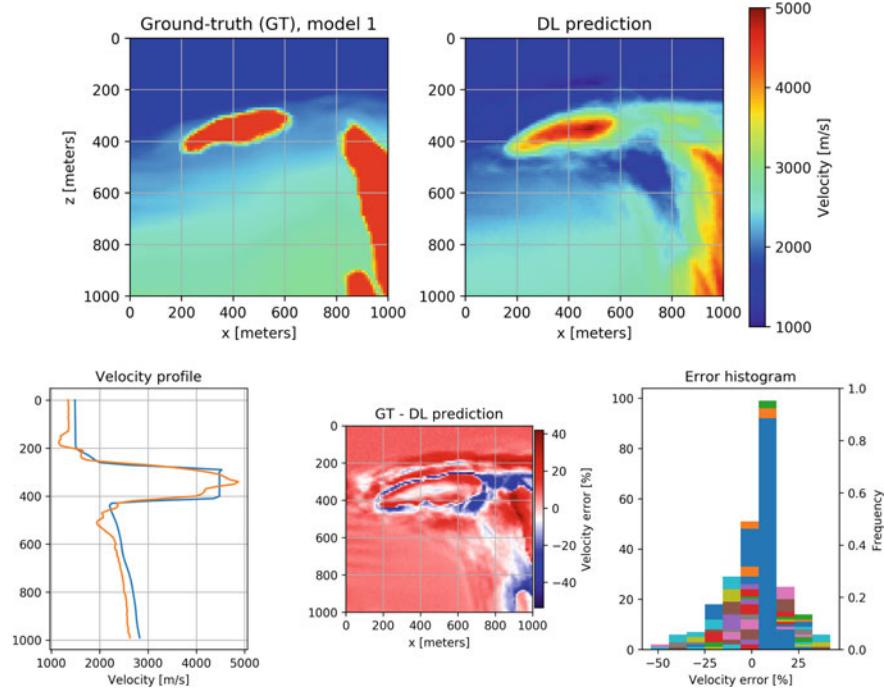
## 6 Feature Extraction-Free Results

Human biased feature extraction is not desired when truly following the deep learning paradigm, which encourages an end-to-end learning process which maps from the relevant elements of the raw data to the ground-truth. After the initial success of the semblance-based approach, experiments were conducted with a modified version of the network (as depicted in Fig. 9) that accepts seismic gathers without manipulation as inputs (Fig. 17, right). The label dataset (Fig. 17, left) is composed by the velocity models as described in previous sections. The main change in the network, compared to the one presented in Sect. 4, is in the input layer. Now the input is the raw seismic shot gathers which are of the dimension (number of shots  $\times$  number of receivers  $\times$  time samples). Each data/label pairing is composed by the later described 3D seismic gather and the corresponding velocity model as label. Furthermore, this network's training used the Nesterov optimizer ADAM with a learning rate of 1e-03, batch size of 20 (per GPGPU) and the experiments where executed for 250 epochs using the MSE loss function. Training takes less than two hours and inference takes only seconds.

It can be observed in Fig. 18 that the velocity model used as label is larger than the ones used in Sect. 4 and much more rich in features (velocity variation), this is because these velocity models belong to datasets used in exploration in the Gulf of Mexico (due to confidentiality reasons actual geographical locations can not be shared). Consequentially, the generated seismic data is much closer to what actual field records look like. The decision of using this data is not random, the final purpose is to expose the ML approach to real field data and therefore cross the threshold from research



**Fig. 17** (left) Example velocity model from the training dataset, (right) examples of corresponding seismic traces (only three selected sources), obtained by wave propagation and without first arrival removal, where vertical axis represents time and horizontal is offset from source location



**Fig. 18** Results examples for a trained model without pre-computed features. (top, left) shows the ground-truth and (top, right) the corresponding prediction. (bottom, left) comparison of the velocity profile for  $x = 500$ , vertical axis represents depth in meters

into industrial-tested tool. One step in data preparation is to downsample the label and data, in particular the data was downsampled to fit in GPGPU memory, which for these experiments are three NVIDIA V100s with 16 GB of internal memory each. The input data dimension used is  $31 \times 256 \times 300$  (where dimension where described in previous paragraph). The label and predicted model dimension is  $100 \times 100$  samples, where the first dimension represents the horizontal axis and the second dimension represents depth or vertical dimension. The total size of the training dataset is 960 samples, where 80 sample were separated for validation and another 80 samples for testing.

The results training no pre-computed features (in Fig. 18) are at least comparable if not superior to the ones presented in Sect. 4. Comparable in the sense that all major features of the expected reconstructed models are present and the error ranges are similar. In quantitative terms, the test dataset SSIM metric is 0.8181 and the  $R^2$  metric is 0.8272. This two figures are slightly less impressive than the ones reported in Sect. 4, three main factors are the culprit: complex velocity models, smaller dataset and forced downsampled data. In qualitative terms, the largest errors appears around the fine-grained contours of salt bodies, which is also the case for the traditional

techniques as is shown in Sect. 5. Nonetheless, the results are superior in the sense that the expected reconstructed models are more detailed and therefore harder to predict, also these velocity models are essentially what is used in exploration geophysics within seismic imaging workflows.

## 7 Conclusions

This chapter presents a novel DL approach to a key geoscience problem [5, 37]. By utilizing DL, it is possible to predict earth models directly from the recorded seismic data. Essentially, we are replacing an nonlinear inverse problem with a data-driven learning process. Results with synthetic data achieve high visual accuracy, both with structural similarity image metric (SSIM) and PSNR. This solution enables fast turnaround of exploration workflows that nowadays take weeks to complete, therefore empowering domain experts allowing them to focus on the most complex prospects within the data. The proposed approach can be extended to other relevant geoscience problems where accurate earth model are also required. Future work is twofold: extension to 3D tomography, namely reconstruction of three dimensional subsurface models, and validation with field recorded seismic data.

**Acknowledgements** The authors would like to thank Shell International Exploration and Productions for supporting and allowing us to share this material.

## References

1. Alpak, F.O., Araya-Polo, M.: Big loop in the machine learning era. In: 81st EAGE Conference and Exhibition (2019)
2. Farmer, P., Miller, D., Pieprzak, A., Rutledge, J., Woods, R.: Exploring the subsalt. *Oilfield Rev.* 50–64 (1996)
3. AlRegib, G., Deriche, M., Long, Z., Di, H., Wang, Z., Alaudah, Y., Shafiq, M.A., Alfarraj, M.: Subsurface structure analysis using computational interpretation and learning: a visual signal processing perspective. *IEEE Signal Process. Mag.* **35**(2), 82–98 (2018)
4. Di, H., Wang, Z., AlRegib, G.: Why using CNN for seismic interpretation? An investigation. In: SEG, pp. 2216–2220 (2018)
5. Rawlinson, N., Pozgay, S., Fishwick, S.: Seismic tomography: a window into deep earth. *Phys. Earth Plan. Inter.* **178**(3), 101–135 (2010)
6. Araya-Polo, M., Dahlke, T., Frogner, C., Zhang, C., Poggio, T., Hohl, D.: Automated fault detection without seismic processing. *Lead. Edge* **36**(3), 208–214 (2017)
7. Araya-Polo, M., Jennings, J., Adler, A., Dahlke, T.: Deep-learning tomography. *Lead. Edge* **37**(1), 58–66 (2018)
8. Claerbout, J.F.: Straightedge Determination of Interval Velocity (1978)
9. Virieux, J., Brossier, R., Mtivier, L., Etienne, V., Operto, S.: Challenges in the full waveform inversion regarding data, model and optimisation. In: 74th EAGE Conference and Exhibition-Workshops (2012)

10. Stork, C., Clayton, R.W.: Linear aspects of tomographic velocity analysis. *Geophysics* **56**(4), 483–495 (1991)
11. Sava, P., Biondi, B.: Wave-equation migration velocity analysis. I. theory. *Geophys. Prospect.* **52**(6), 593–606 (2004)
12. Tromp, J., Tape, C., Liu, Q.: Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels. *Geophys. J. Int.* **160**(1), 195–216 (2005)
13. Fomel, S.: Traveltime computation with the linearized Eikonal equation. *Stanf. Explor. Project Rep.* **94**, 123–131 (1997)
14. Jin, K.H., McCann, M.T., Froustey, E., Unser, M.: Deep convolutional neural network for inverse problems in imaging. *IEEE Trans. Image Process.* **26**(9), 4509–4522 (2017)
15. Lucas, A., Iliadis, M., Molina, R., Katsaggelos, A.K.: Using deep neural networks for inverse problems in imaging: beyond analytical methods. *IEEE Signal Process. Mag.* **35**(1), 20–36 (2018)
16. McCann, M.T., Jin, K.H., Unser, M.: Convolutional neural networks for inverse problems in imaging: a review. *IEEE Signal Process. Mag.* **34**(6), 85–95 (2017)
17. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press. <http://www.deeplearningbook.org> (2016)
18. Hastie, T., Tibshirani, R., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics. Springer, New York (2009)
19. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. In: *The Handbook Of Brain Theory and Neural Networks*, vol. 3361, no. 10 (1995)
20. Couchot, J.F., Couturier, R., Guyeux, C., Salomon, M.: Steganalysis via a convolutional neural network using large convolution filters (2016). CoRR, [arXiv:1605.07946](https://arxiv.org/abs/1605.07946)
21. Kinga, D., Ba, J.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations (ICLR), vol. 5 (2015)
22. Dozat, T.: Incorporating nesterov momentum into adam. In: Proceedings of ICLR Workshop (2016)
23. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). Software available from [www.tensorflow.org](http://www.tensorflow.org)
24. Chollet, F., et al.: Keras. <https://keras.io> (2015)
25. nVIDIA, Tesla K40 and K80 GPU accelerators for servers. <http://www.nvidia.com/object/tesla-servers.html> (2014)
26. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* **13**(4), 600–612 (2004)
27. Červený, V.: Seismic Ray Method. Cambridge University Press (2001)
28. Vidale, J.: Finite-difference calculations of travel times. *Bull. Seismol. Soc. Am.* **78**(6), 2062–2076 (1988)
29. Ellefsen, K.J.: A comparison of phase inversion and traveltime tomography for processing near-surface refraction traveltimes. *Geophysics* **74**(6), WCB11–WCB24 (2009)
30. Tarantola, A.: Inversion of seismic reflection data in the acoustic approximation. *Geophysics* **49**(8), 1259–1266 (1984)
31. Romdhane, A., Grandjean, G., Brossier, R., Réjiba, F., Operto, S., Virieux, J.: Shallow-structure characterization by 2D elastic full-waveform inversion. *Geophysics* **76**(3), R81–R93 (2011)
32. Virieux, J., Operto, S.: An overview of full-waveform inversion in exploration geophysics. *Geophysics* **74**(6), WCC1–WCC26 (2009)
33. Aki, K., Richards, P.G.: Quantitative seismology
34. Aster, R.C., Borchers, B., Thurber, C.H.: Parameter Estimation and Inverse Problems. Elsevier (2005)

35. Bunks, C., Saleck, F.M., Zaleski, S., Chavent, G.: Multiscale seismic waveform inversion. *Geophysics* **60**(5), 1457–1473 (1995)
36. Farris, S.: Tomography: a deep learning vs full-waveform inversion comparison. In: First EAGE Workshop on High Performance Computing for Upstream in Latin America (2018)
37. Taner, M.T., Koehler, F.: Velocity spectra—digital computer derivation applications of velocity functions. *Geophysics* **34**(6), 859–881 (1969)

# Traffic Light and Vehicle Signal Recognition with High Dynamic Range Imaging and Deep Learning



Jian-Gang Wang and Lu-Bing Zhou

**Abstract** Use of autonomous vehicles aims to eventually reduce the number of motor vehicle fatalities caused by humans. Deep learning plays an important role in making this possible because it can leverage the huge amount of training data that comes from autonomous car sensors. Automatic recognition of traffic light and vehicle signal is a perception module critical to autonomous vehicles because a deadly car accident could happen if a vehicle fails to follow traffic lights or vehicle signals. A practical Traffic Light Recognition (TLR) or Vehicle Signal Recognition (VSR) faces some challenges, including varying illumination conditions, false positives and long computation time. In this chapter, we propose a novel approach to recognize Traffic Light (TL) and Vehicle Signal (VS) with high dynamic range imaging and deep learning in real-time. Different from existing approaches which use only bright images, we use both high exposure/bright and low exposure/dark images provided by a high dynamic range camera. TL candidates can be detected robustly from low exposure/dark frames because they have a clean dark background. The TL candidates on the consecutive high exposure/bright frames are then classified accurately using a convolutional neural network. The dual-channel mechanism can achieve promising results because it uses undistorted color and shape information of low exposure/dark frames as well as rich texture of high exposure/bright frames. Furthermore, the TLR performance is boosted by incorporating a temporal trajectory tracking method. To speed up the process, a region of interest is generated to reduce the search regions for the TL candidates. The experimental results on a large dual-channel database

---

This work was supported by the A\*STAR Grant for Autonomous Systems Project, Singapore. Both authors contributed equally to this work. This work was done when Lu-Bing Zhou was with Institute for Infocomm Research. He is now with nuTonomy, Singapore 139954.

---

J.-G. Wang (✉)

Robotics Department, Institute for Infocomm Research, 1 Fusionopolis Way, #21-01 Connexis (South Tower), Singapore 138632, Singapore  
e-mail: [jgwang@i2r.a-star.edu.sg](mailto:jgwang@i2r.a-star.edu.sg)

L.-B. Zhou

Autonomous Vehicle Department, Institute for Infocomm Research, 1 Fusionopolis Way, #21-01 Connexis (South Tower), Singapore 138632, Singapore  
e-mail: [zlbining@gmail.com](mailto:zlbining@gmail.com)

have shown that our dual-channel approach outperforms the state of the art which uses only bright images. Encouraged by the promising performance of the TLR, we extend the dual-channel approach to vehicle signal recognition. The algorithm reported in this chapter has been integrated into our autonomous vehicle via Data Distribute Service (DDS) and works robustly in real roads.

**Keywords** Traffic light recognition · Vehicle signal recognition · High dynamic range imaging · Deep learning · Autonomous vehicle · Data distribute service

## 1 Introduction

Traffic Light Recognition (TLR) locates the traffic light from an image and then estimates the status of the light signal. Vehicle Signal Recognition (VSR) estimates the signal of the vehicles ahead from an image. Automatic recognition of traffic light and vehicle signal are two of perception (functionalities) for Advanced Driver Assistance Systems (ADAS) or Autonomous Vehicle (AV) because failure of following traffic light or vehicle signal could lead to a fatal accident. There have been lots of studies on TLR. However, not much attention has been paid to practical TLR problems. The most challenging issues of TLR include computation time, day/night lighting conditions, confusion of tail lights or other kinds of ambient light, low image resolution and vehicle occlusion. Most of existing TLR approaches are sensitive to lighting conditions because only bright images are used. In this chapter, in the premise of ensuring real-time conditions, we are interested in TLR problem under varying lighting conditions and confusion of tail lights. A two-stage approach is proposed to solve the problems: detect traffic light candidates in low exposure/dark image which is less sensitive to lighting conditions and then recognize their traffic light state in high exposure/bright image which has rich texture. Deep learning is adopted to improve the recognition accuracy significantly.

Some good surveys on TLR can be found in [1–3]. The existing TLR methods can be roughly divided into three categories: (1) template matching; (2) circular extraction; and (3) color distribution. In the first category, templates of red or green light are matched with the extracted regions. The circular shape is detected from images by using Hough transform in the second category. The third category is mainly color segmentation. One of the major disadvantages of these three categories approaches is the high sensitivity to lighting conditions. Color and shape information are used to detect TL candidates [4–7]. Some image preprocessing is applied to prune the candidates before being fed to the classifier. Before pruning candidates using shape, temporal, edge and symmetry, image segmentation in HSV [8] or RGB [9] space is adopted. In order to recognize the traffic light states robustly, an adaptive template matching method is proposed [10].

In order to improve detection accuracy, region of interest (ROI) is used to reduce the search region. Map and GPS (annotated) are used to generate ROI [11, 12]. Some

non-passive approaches, e.g. vehicle-to-light or car-to-car [13–15] are also proposed. However, they are not widely adopted because special infrastructures are required.

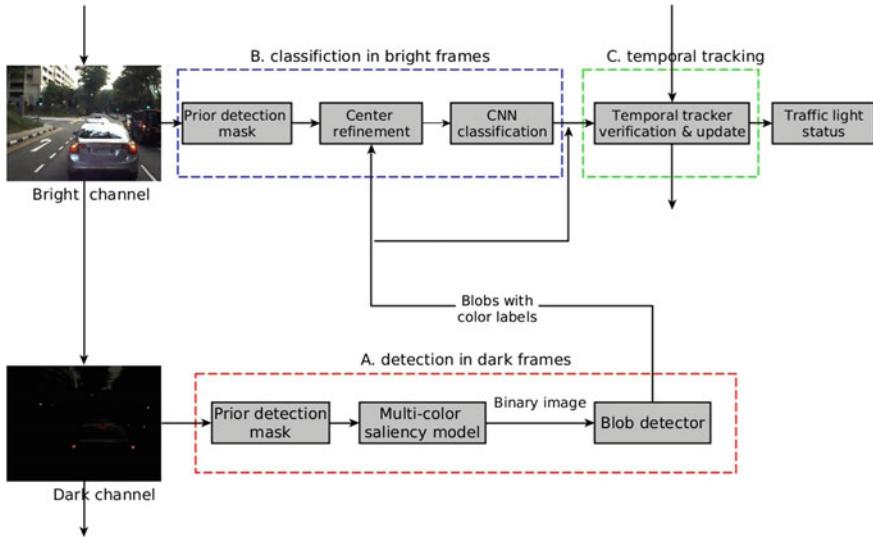
High Dynamic Range (HDR) imaging is able to reproduce images with greater range of illuminance than traditional imaging technology. This can be done by capturing and then combining several different, narrower range, exposures of the same subject matter [16]. Traditional cameras capture images with a limited exposure range, referred to as LDR. Compared to HDR, the LDR results in the loss of detail in highlights or shadows.

We use HDR in a different way than existing HDR research. Instead of generating HDR images from multiple exposure images, we use images at different range of illumination levels independently. A dual-channel method is proposed to detect and recognize traffic lights and vehicle signals. The low and high exposure images are used to detect light and recognize light status, respectively.

We note that the detection of lights from a lower exposure/dark image is much more robust than that from a high exposure/bright image (sensitive to environment illumination). In this chapter, a HDR camera, which can provide more than one image with different exposures, is used. As far as we know, this is the first time HDR camera is used in traffic light or vehicle signal recognition. The authors [17] used two channels by fusing simple color thresholding segmentation and SVM [18] (Histogram of Gradient features are used). They conducted some experiments on urban scene images but have not reported accuracies in their paper. As we know, the thresholding color segmentation is very sensitive to the outdoor illumination. Compared to deep learning features [19], hand-crafted features, like HOG, cannot expect higher accuracy even if more training samples are provided. In this chapter, we extensively investigate the HDR imaging and deep learning with the applications in TLR and VLR. The TL candidate detection is executed with a low exposure/dark image which is fast and robust to the environment illumination. Once TL candidates are found, the TL states can be identified by applying machine learning algorithm, e.g. Adaboost [4]. Deep learning is a state of the art machine learning methods. The advantage of deep learning over traditional technologies is the accuracy is not saturated with growing number of training samples. It has been found that the accuracy of the traditional machine learning, e.g. SVM, will be not improved even when more training samples are added. In addition, deep learning has applied widely as the computer hardware progress makes it feasible for real-time applications. In this chapter, given a TL candidate region, their counterpart in bright image is passed to a Convolutional Neural Network (CNN) to identify the traffic state. Furthermore, the TLR accuracy is improved by incorporating a tracking technology.

The traffic light and vehicle signal lights recognition [4, 20, 21] is essentially a computer vision problem. Machine learning could be enhanced by some preprocessing or post processing, e.g. image processing or geometric estimation. As the normal road is flat, the number of the TL candidates could be reduced by some geometric properties, e.g. the perspective projection between the camera and the road. This can save computational cost significantly because each candidate needs to be classified.

Figure 1 shows the diagram of our traffic light recognition system. HDR camera makes it possible to set different parameters for different channels to cover dynamic



**Fig. 1** Diagram of the traffic light recognition system [22]

range. To speed up the recognition processing, the candidates are pruned by saliency map and region of interest (ROI). The ROI in our approach is determined by: (1) calibrating the camera with respect to the ground world coordinate; (2) the knowledge about the physical heights of the TLs. Finally, based on temporal trajectory analysis, we develop a tracking technology. In doing so, the robustness and accuracy of the TLR have been improved.

This chapter is a technical summary of three papers [20, 22, 23] which respectively uses related technologies for TLR and VSR. The original methodologies can refer to these papers.

The remainder of the chapter is organized as follows. The HDR imaging based traffic light detection will be discussed in Sect. 2. The CNN traffic light recognition is discussed in Sect. 3. In Sect. 4, tracking technology is discussed. The experimental results are given in Sect. 5. The extension of the dual-channel method to vehicle signal recognition is discussed in Sect. 6. Conclusion and future work are discussed in Sect. 7.

## 2 HDR Imaging Traffic Light Detection

### 2.1 *HDR Imaging*

High Dynamic Range (HDR) imaging is the compositing and tone-mapping of images to extend the dynamic range beyond the native capability of the capturing

device [16]. HDR technology has been successfully applied in both photo and TV to make images/frames have a greater contrast between bright and dark. Different from the existing applications of HDR which enhance visual quality by combining bright and dark images, we use dual-channel separately and combine them using the association between the two channels.

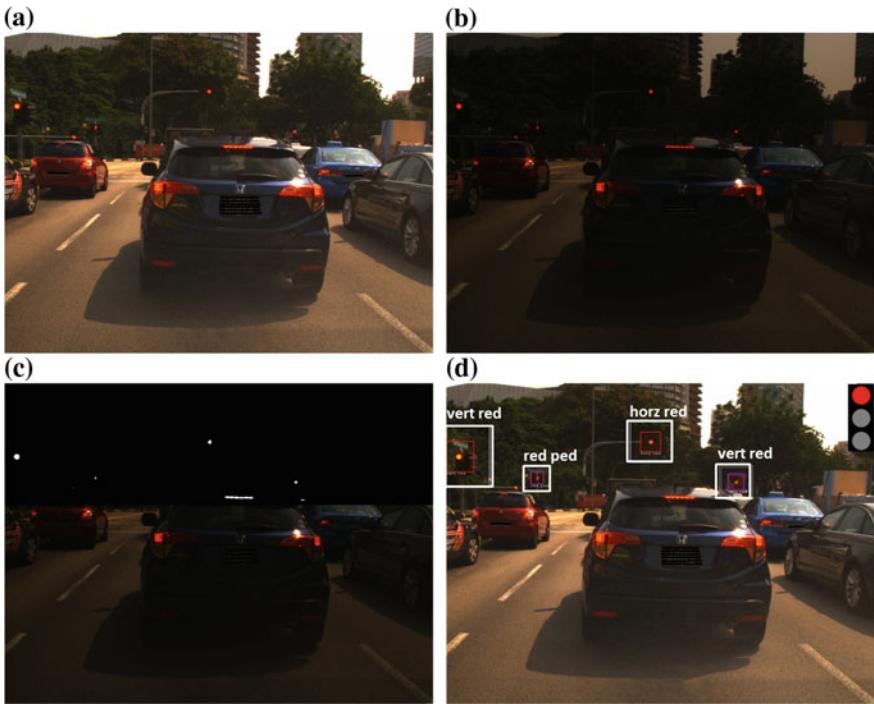
The motivation for us to adopt HDR is that the higher detection ability of the dark images and higher recognition ability of the bright images. Although the dark and bright image are not exact captured simultaneously, the relatively time difference between them is short enough to be neglected. In other words, we can easily find the corresponding regions between bright and successive dark images, and vice versa. This helps us associate the detected traffic light candidates in the dark image with their location in the bright image.

As mentioned, the dark images are used for traffic light candidate detection and bright images are used for recognition. As for vehicle signal recognition, the bright images are used for vehicle detection and dark images are used for vehicle signal recognition.

## 2.2 *Dark Images for Detecting Traffic Light Candidates*

It is an essential step to detect TL candidates from images for a successful traffic light state classification/tracking system. Currently, most of TLR systems detect traffic light using only bright images. However, similar to other detection problems, their performance is very sensitive to the environment lighting conditions, and confusion with the tail lights of vehicle ahead or other similar ambient light, for example, traffic sign, temporary roadblocks, pedestrian. How to robustly detect traffic light candidates under varying illumination is still an open problem. In this chapter, instead of using a single image, we propose a novel method to use dual channel (low and high exposure, respectively) provided by a HDR camera. Unlike previous HDR imaging in which a single image is synthesized from bright and dark channels, in our approach these channels are used separately to detect and recognize traffic lights. As mentioned in Sect. 2.1, a HDR camera, with more dynamic ranges than a normal camera, can be used in such a way that the successive two channels can be set as high exposure and low exposure, respectively. The traffic light candidates detected in a dark image can be located easily in bright channel as they are captured within a very short time interval, about 40 ms for our camera having 25 fps with high-definition serial digital interface (HD-SDI). The association between the candidate regions on dark and bright images is not affected largely by high speed moving. Anyway, a way to re-locate the TL candidate detection results on the bright image is proposed in this chapter, and will be discussed in Sect. 3.1.

The way we use the HDR imaging makes the traffic light candidate detection more robust than others because the lights are with a clean dark background on low exposure images. By using this HDR dual-channel mechanism, undistorted color and



**Fig. 2** Traffic light detection and recognition with dual-channel mechanism [22]. **a** High exposure/bright image; **b** Low exposure/dark image; **c** Dark image with saliency map of the ROI; **d** Traffic lights candidate detection and recognition results. The traffic light state result is displayed in the upper right

shape information on a dark image and rich context information on a bright image are fully used.

Figure 2 shows an example of the dual-channel TLR. We can see that the lights, including traffic lights and vehicles' tail lights, are prominent in the dark image. The rich context can be seen from the bright image.

Low lighting conditions is a challenging issue in using HDR to recognize traffic light. Traffic light candidates are detected from dark image by a simple color thresholding segmentation in [17]. The detection performance could be unreliable as it is hard to adapt the varying illumination conditions with a threshold. A saliency map filtering, aims to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze [24], is adopted in this chapter to handle low lighting problem.

## 2.3 Saliency Map Filtering

Most of the existing traffic light recognition methods detect traffic lights (color blobs) by tuning thresholds. The color information is used for locating and identifying traffic light states. YCbCr [25], instead of RGB, is considered for this purpose because the color and intensity are mixed in three channels of the RGB color space.

Usually, the parameters used to identify the traffic light states (red, green and amber) are very sensitive to environment lighting conditions. As verification needs to be done for each pixel in order to determine the state, the time consumption is linearly increased with the number of colors. In order to speed up the process, a non-parameter model is proposed in this chapter to extract blobs of various colors simultaneously. RGB color space is used in this chapter in order to illustrate the robustness of our method although the performance could be better when HSV or other spaces is adopted [17].

Our method contains followings steps. Firstly, the 3D RGB color space is divided into grids,  $M \times M \times M$ .  $M$  is set to be 32 (without fine tuning) in this chapter. Secondly, the histograms for each state, including red, green and amber colors, are calculated from samples. Let's define normalized histogram  $[0, 1]$  for red, green and amber as  $H_r$ ,  $H_g$  and ..., respectively. Those values above 0.1 in  $H_r$ ,  $H_g$  and  $H_a$  are truncated to prevent extreme dominance of a single color bin. The resulting histograms are renormalized to  $[0, 1]$ . Given an input image,  $I$ , the saliency score of a pixel  $(i, j)$  in red channel is computed as

$$S_r(i, j) = \sum_{(i', j') \in N_d(i, j)} H_r(i', j') \quad (1)$$

where  $N_d(i, j)$  represents neighborhood of pixel  $(i, j)$  within a maximal distance of  $d$ . A saliency mask can be obtained by applying a threshold,  $T$ , to  $S_r$ . In this chapter,  $d$  is set to be 2 and  $T$  is set to be 0.2. No fine tuning is done for these settings. Although saliency maps can be computed individually with the histogram models for different light types, it is computationally redundant to compute the saliency score of each pixel for each color. In our approach, a *Max* operator is proposed to combine the histograms of three traffic light states (red, green and amber):

$$H = \max(H_r, H_g, H_a) \quad (2)$$

A final saliency map,  $S$ , is obtained by replacing  $H_r$  in Eq. (1) with  $H$ .

$$S(i, j) = \sum_{(i', j') \in N_d(i, j)} H(i', j') \quad (3)$$

If the saliency value of a pixel is found to be above the threshold, then the channel saliency scores will be re-computed using the three channel histogram models. The pixel is assigned to a type which is with the maximize channel saliency score.



**Fig. 3** Saliency map [22]. **a** High exposure/bright image; **b** Low exposure/dark image; **c** Saliency map of (b); **d** Saliency map with color label

Following the way mentioned-above, most of the pixels could be filtered out by the final saliency score, and the types of other pixels could be determined by individual saliency models. Figure 3 shows an example of the proposed saliency model.

Function `findContours()` in the OpenCV [26] is used to extract contours of the blobs from the resulting binary image. Some obvious incorrect blobs can optionally be removed based on shape analysis, e.g. area or circularity.

## 2.4 Auto Exposure for Uncontrolled Illumination

Although many researchers have devoted research to lighting problems in computer vision, it is still a challenge for a vision system to work robustly under varying light conditions. As mentioned above, adjusting camera exposure could be an efficient way to detect traffic light. Zebra2 (2.8 MP Color GigE Vision) [27], the HDR camera used in this chapter, provides auto-exposure. However, this function will be disabled when high dynamic range setting is activated. TL candidate detection could be unstable because for a real scene, e.g. sunlight and skylight, dynamic range may be wider than that for the camera setting. The severe illumination changes under uncontrolled outdoor environment have to be considered although saliency map could make the detection under large illumination variations more reliable than a simple threshold. Auto exposure, i.e. automatically adjusting exposure parameters, e.g. gain or shutter speed, should be helpful to keep image features. Here, auto exposure for dual channel will be considered.

Auto exposure has been investigated in literature [28–30]. However, we need a fast auto exposure approach for our autonomous vehicle application. In this chapter, we propose a real-time auto exposure approach. The exposure is adjusted by observing the difference between the mean intensity of an image mask and a reference value.

Let  $I_t$  represents an expected mean image intensity,  $I_c$  represents the mean intensity of the current frame, a factor is defined as

$$f = \frac{I_t}{I_c} \quad (4)$$

Our objective is to let  $f$  in Eq. (4) tend to 1, i.e.  $I_c$  tends to  $I_t$ , by updating gain or shutter.

To obtain expected  $f$ , shutter and gain are jointly adjusted within their respective ranges  $[s_{\min}, s_{\max}]$  and  $[g_{\min}, g_{\max}]$ . In actual implementation, the shutter is adjusted before gain, as noise could come together with a large gain value. The adjustment of shutter will result in a factor:

$$f_s = \frac{s_t}{s_c} \quad (5)$$

where  $s_c$  represents current shutter value,  $s_t$  and represents the updated shutter value. It is known that the shutter value is directly proportional to intensity. If desired factor  $f$  can be achieved by only adjusting shutter within its range, i.e.  $f_s = f$ , then no gain adjustment is needed. However, if  $f_s$  cannot lead to a targeted image intensity, then the shutter will be updated to its extreme within the range, and the gain will be adjusted to cover the remaining portion of the factor, i.e.  $f = f_s f_g$ , where  $f_g$  is a gain factor. The remaining factor can be achieved easily by adjusting gain based on a common observation: when the gain adjustment (increase or decrease) approximately 6 db, the intensity doubles or halves.

## 2.5 Region of Interest (ROI)

Real-time is a requirement for a practical TLR system. One way to speed up TL detection is to have some prior knowledge about the traffic lights' location on an image. By calibrating the camera with respect to 3D world, a region can be determined on an image which could contain traffic light candidates.

The prospective projection of a camera can be represented as a transform matrix. The relationship between the 3D world and 2D image can be represented as a transformation matrix:

$$\begin{bmatrix} ut \\ vt \\ t \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (6)$$

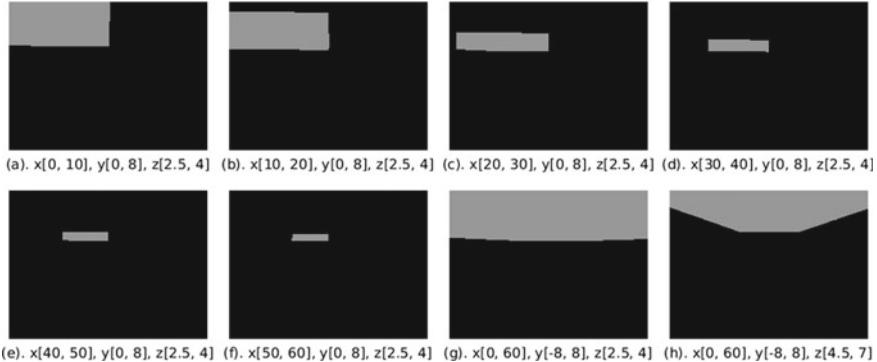
where  $(x, y, z)$  and  $(u, v)$  represent world coordinates and image coordinates, respectively. In this chapter, the world coordinate system is defined XY lies on the ground and the Z-axis is upward and perpendicular to XY. The origin of the XYZ corresponding to the frontal middle point of the host vehicle; the X-axis towards the front of the host vehicle. The Y-axis towards the left to make the XYZ follows right-hand rule. The calibration is a process to estimate the eleven unknown parameters,  $a$ , in  $3 \times 4$  matrix of Eq. (6). At least four groups of 3D and 2D coordinates are needed for this purpose. In practical, more than four groups of 3D and 2D coordinates can be used to calibrate the camera. The eleven parameters are estimated from these groups of data by solving a least-square fitting problem. In this chapter, these groups of data are obtained by reading the coordinates of a few known-size calibration objects.

The location of the TLs on an image could be estimated when the vehicle's localization and 3D localization of TLs on the map are given. However, high accuracy map and localization estimation are needed for this kind of method, makes it hard to be adopted in real practical application. In this chapter, instead, accurate map and vehicle localization, the region of interest (ROI) method is adopted to speed up the TL detection.

In our experiments, we have not made any assumptions about map, traffic lights' location or host vehicle's pose. When no localization information is provided, a roughly ranges in x, y, z direction with a very roughly 2D GPS position is still useful. Based on these 3D ranges, an intensive 3D grid can be made and 2D image ROI can be correspondingly generated. In other words, a ROI, corresponding to the longest distance, is adopted where the traffic lights candidates can be found.

In this chapter, the detection range in XYZ for a vertically hanged traffic light is defined as follows. X (longitudinal): [0 m, 70 m]; Y (lateral): [-8 m, 8 m] and Z (upward): [2.5 m, 4 m]. These parameters are set based on normal traffic light cases in Singapore. To estimate possible ROI,  $(x, y, z)$  may change within range. Figure 4g and h show two detection masks or ROI for horizontally hanged TLs, obtained by changing Z within [4.5 m, 7 m]. In the case that either vehicle pose or TLs location is accessible, such ROI could be further shrunk. More examples for ROIs corresponding to different ranges are shown in Fig. 4a-f. Figure 4g is used as the ROI.

With the help of the ROI, the computation cost for detecting traffic light candidates can be reduced significantly. We can see this from the experimental results to be discussed in Sect. 5. Low computation cost is very important for real-time applications, such as autonomous vehicles, which need to run a few models, e.g. perception, navigation, simultaneously. Besides time saving, the traffic light recognition accuracy has been improved since some false positives located outside the ROI can be prevented.



**Fig. 4** Eight ROIs for different x, y and z [22]. The ROI in (g) is used in our real experiments

### 3 Traffic Light Recognition with Deep Learning

Thanks to large data available and hardware progress, deep learning, as a state of the art machine learning technology, has achieved very promising results in computer vision (e.g. object detection, data augmentation), speech recognition and natural language etc. [31]. Hierarchical representations of training data rather than hand-crafted features can be learned by a deep architecture.

In this chapter, we adopt deep learning to recognize TL status from images. Similar to other deep learning applications, the idea in this chapter is that a convolutional neural network (CNN) is able to classify a TL candidate into a TL state efficiently. In this chapter, we have shown that it is possible to develop a real-time high accuracy TLR system when a deep model as well as parameters are designed carefully.

As we discussed in Sect. 2, the location of the traffic light candidate on the bright image can be determined by their locations on the dark images because the interval time between the successive bright/dark frames is very short and can be neglected. First of all, we will discuss the correspondence between the bright and dark channels in the next section.

#### 3.1 Dual Channel Mechanism

As we know, the two images that captured via low exposure and high exposure channels are not synchronized, i.e. they are not captured simultaneously, although the interval between the two channels' timestamps is very short. The vehicle's motion makes it hard to align the TL candidate detected from dark image with a bright image, especially when the vehicle's vibration (due to movement) cannot be ignored. Hence, a way is needed to re-locate the TL candidate detection results on the bright image.

With the detected candidates on the dark frame, we aim to find the corresponding regions on the next bright frame which is with richer texture. Considering the time

interval between the consecutive frames, new region centre on the bright image could be needed to ensure the regions cropped from the bright image are corresponding to the TL candidates. In this chapter, the center position,  $p$ , and radius,  $r$ , of the TL candidates could be used to estimate the new center on the following bright frame. In details, the new centre is searched within a window,  $12r \times 12r$  in this chapter, centered at  $p$ . The centres of the TL candidates is normally with the highest brightness value and color variance among the pixels within the window. For RGB space, brightness image,  $I$ , is computed as

$$I = 0.2126 * R + 0.7152 * G + 0.07221 * B \quad (7)$$

and the variance image,  $V$ , is computed as

$$V = |R - I| + |G - I| + |B - I| \quad (8)$$

A new center can be found as the highest response in a weighted sum image [32, 33]:

$$\alpha V + (1 - \alpha)I \quad (9)$$

where  $\alpha$  is a weight. As the brightness changes significantly when the lighting conditions changes,  $\alpha$  is set to be 0.7 in this chapter. As mentioned above, a  $12r \times 12r$  window centered at each new center, is cropped from bright frames and used as candidate regions for TL state classification.

### 3.2 Customized Convolutional Neural Network

False positives could be possible, e.g. caused by braking light of the vehicles ahead or other objects with color similar to TL, although most of them can be removed during TL candidate detection stage. In order to improve robustness furtherly, a CNN classifier is applied to identify true positives from false positives.

The accuracy and speed are two considerations for us to select a CNN classifier. As one of the perception models runs in autonomous vehicle, the running speed is an important issue for selecting deep learning model because it will share limited resource with other models, e.g. object detection, lane detection, etc. CaffeNet [34], see Fig. 6, is a 1-GPU version of AlexNet [35] in which the two paths in AlexNet are combined to become one path. A customized version CNN model, similar to CaffeNet [34], is adopted in this chapter. The number of output layer (last layer) is defined as 13, i.e. twelve positive classes and one background class. The positive classes are defined based on possible traffic light types.

- (1) HARL Horizontally Aligned Red Light
- (2) VARL Vertically Aligned Red Light

- (3) HAGL Horizontally Aligned Green Light
- (4) VAGL Vertically Aligned Green Light
- (5) LVL Left Vehicle Light
- (6) RVL Right Vehicle Light
- (7) GAL Green Arrow Light
- (8) RAL Red Arrow Light
- (9) AL Amber Light
- (10) GPL Green Pedestrian Light
- (11) RPL Red Pedestrian Light
- (12) OFRL Other Fake Red Light

Figure 5 shows the annotation results on an image where the true positives are labeled in blue and false positives are labeled in red.

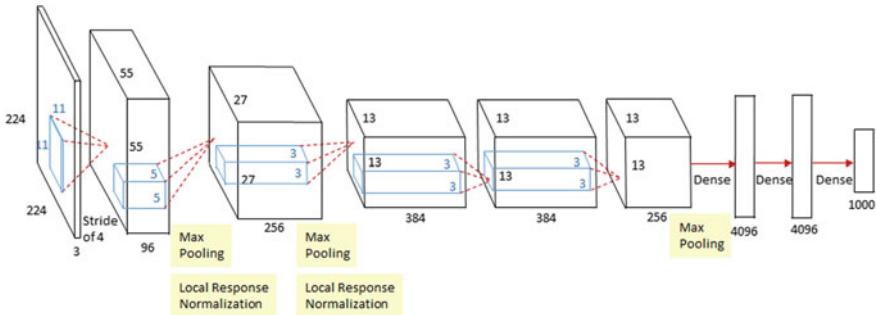
The within-class variance can be reduced by above definitions. For example, by defining horizontal and vertical lights, red traffic lights can be distinguished from false positives, e.g. LVL, RVL and OFRL. Another advantage for this definition is the reduced effort in data collection and annotation.

CaffeNet contains eight layers. The first five layers are conventional layers which transforms one volume of activations to another through a differentiable function. The convolutional layer's parameters consist of a set of learnable filters (convolutional kernels). Each filter is small spatially (along width and height), but extends through the full depth of the input volume. Two processes: max pooling and local response normalization, are added in the first and second layers. The last three layers are fully-connected layers. According to the architecture in Fig. 6, there are 60 million parameters need to be trained.

The CNN classifier's weights are trained with fine tuning strategy. The parameters needed in the training procedure are set based on experiments. In our approach, the basic learning rate and decay of the CNN model are set to be 0.001 and 0.0005,

**Fig. 5** Annotate TL samples on an image [22]





**Fig. 6** The architecture of CaffeNet

respectively. For modified layers, the multipliers of learning rate, i.e. first convolutional layer and output layer, are set to be 10 in the first 2000 iterations, and 1 for the other layers. In total, 50000 iterations are taken in the training procedure.

## 4 Temporal Trajectory Analysis

In this chapter, instead of using normal tracking technologies, such as Kalman filter or particle filter, we propose a simple but efficient temporal trajectory analysis for improving accuracy and robustness of traffic light recognition.

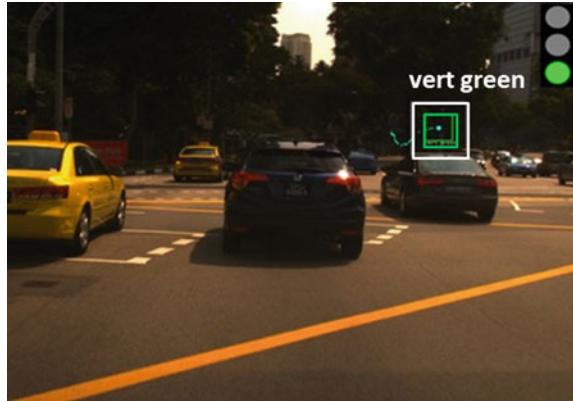
As mentioned in Sect. 2.4, a HDR camera, Zebra2 [27], is used in this chapter. This high speed camera ensures that the targets found on images constantly change from frame to frame. In this scenario, temporal spatial analysis, a process to examine if a target detected in the current frame has ever been found in the nearly same area of the last frames, can be used to track the targets. The traffic is controlled by keeping traffic light status for a certain period of time. As a result, the regions of the light are spatially continuous on the image sequence no matter that vehicles are moving or keeping static. Based on these observations, traffic light recognition can be benefited by proper temporal spatial tracking in two aspects: (1) smoothness is improved as missing or low confident traffic light status could be filled up; (2) isolated false positives could be removed.

We define trajectory as the history of a traffic light instance. A trajectory consists of several components:

- (1) type;
- (2) history locations;
- (3) lifetime;
- (4) discontinuity.

The trajectory is grouped according to light status and stability. Hence, six types of trajectories are defined:

**Fig. 7** The trajectory of a vertically green light (marked as dots in green) is plotted onto one frame [22]



- (1) stable red;
- (2) stable green;
- (3) stable amber;
- (4) temporary red;
- (5) temporary green;
- (6) temporary amber.

For example, “stable red” refers to the trajectory is confirmed as red traffic light status (either horizontally or vertically red light). Another item, lifetime, depicts the period of the trajectory since the first instance is detected. An example of the trajectory analysis is shown in Fig. 7. The trajectory of a vertically green light is plotted onto one frame.

The trajectory pool is updated continuously. At the very beginning, once a new target is found, a temporary trajectory will be initialized for it. To update a temporary trajectory to a stable trajectory, a minimal lifetime (one second in our experiments) and a minimal number of instance (five in our experiments) are required. It should be noted that these parameters are set based on our experiments. A trajectory will be deleted from the pool if its lifetime is longer than a threshold (seventy seconds in our experiments). In the current traffic light control system, the lifetime of red, green or amber lights is below this threshold. Sometimes, the red light could last longer than this threshold, it should be feasible to split the history into two trajectories in this case. When a new frame is given, traffic lights are detected. The new targets are then added to the pool. Assume a red light status is recognized from a frame, those trajectories in the pool having red light will be checked. The new location will be added into a trajectory if the distance from the new location to the trajectory is minimum among the red trajectories in the pool and below a pre-defined threshold (sixty-pixel in our experiments). A new temporary red trajectory will be created starting from this new target if no such trajectory can be found. The new target becomes a stable red light if a stable trajectory is found. Otherwise, the new target (could be a false positive) is recorded as a temporary red light.

## 5 Experimental Results

In this section, the quantity analysis (in terms of precision and recall) of the proposed method on a large database will be conducted. The comparison with the state of the art is provided to show the advantages of our approach. The experiments on both database and real roads have shown that our TLR method satisfies the speed and accuracy requirements of an autonomous vehicle.

### 5.1 Evaluation of Performance

One of the standard performance evaluation methods is precision and recall. They are defined as Eqs. (10) and (11).

$$\text{Precision} = \frac{TP}{TP + FP} \quad (10)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (11)$$

where  $TP$  represents number of the True Positive samples,  $FP$  represents the number of the False Positive samples and  $FN$  represents the number of the False Negative samples.

In this section, the quantitative analysis of our method in terms of precision and recall is conducted. For this purpose, a large database has been collected using our autonomous vehicle. The number of the true positives, false positives and false negatives are computed for computing precision and recall with Eqs. (10) and (11).

The database contains 4,142 images. The images are selected such that each class contains nearly same number of samples. A total of 21,070 boxes were manually annotated on these images. There are about 1,750 boxes for each class. The training set consists of 3,722 (about 90%) images. The evaluation set contains 420 images. In order to train network, we generate about one million samples from above seed samples. The scale and translation technology is adopted for generating new samples from seed sample. Although there is no special requirement about the generation of the training samples, during the experiments, we found that the performance is affected by the balance among the number of samples for each class. The generation of the new samples is presented as follows.

In this chapter, the resolution of the original image is  $1600 \times 1200$ . The uniform random distribution is adopted for shifting and scaling original TL region to generate new training samples. The region center of the traffic light candidate are shifted from  $-0.2$  to  $0.2$  times of the candidate rectangle's width or height, and then resizing the region from 1 to 1.2 times. The new samples are finally resized to  $111 \times 111$ .

To evaluate our system, the algorithm runs over 63 new video sequences (each video about 4 min long). The testing videos contain the samples under different

conditions: day time, weather, express way and urban road. 1,800 images, sampling interval 80 frames, are selected from above video sequences. The ground truth comes from these images contains 5,229 samples. Considering the ROI, there are 3,099 samples.

Table 1 gives the experimental results where the test results with the ROI are recorded in brackets.

We can see from Table 1 that the vehicle signal light recognition accuracy is worse than that of other classes. The reason could be that there are not enough vehicle signal lights samples compared with the one of traffic light samples. Another reason for this could be that the vehicle lights' type are much more than that of others. We have to collect more training samples to cover more kinds of vehicle lights if we want to improve vehicle signal light recognition accuracy. Anyway, by applying the ROI discussed in Sect. 2.5, most of the vehicle lights will be removed from the results because they are at the lower part of the images (see Table 1).

Based on the results given in Table 1, the precision and recall are obtained and the results are given in Table 2. From Table 2, we can see that the accuracies of the average recall and precision are improved from 98.04% to 99.03% and from 97.45 to 98.91%, respectively. The detection rate of traffic light candidate is computed as follows.

$$D = (TP + FN)/G \quad (12)$$

where  $G$  represents the number of the ground truth.

The ground truth for each class is given as the first row in Table 3. Based on Table 1 and Eq. (12), the detection rates are computed and listed in Table 3. The average detection rate is improved from 96% to 97.85 when ROI is applied, see Table 3.

## 5.2 Comparison with State of the Art Algorithms

In order to prove the advantages of our approach, the performances achieved by our approach and that of the state-of-the-art are compared. To the best of our knowledge, no publically available HDR TLR benchmark database can be found for such comparison. Most published TLR systems evaluate their method on their own databases collected using a single color camera. One exception we can find from literature is [17] which use multiple exposure images. They conducted experiments on several urban scenes, but there was no accuracy reported.

Nevertheless, the performances achieved by our approach and the state of the art deep learning object detection approach are compared in this chapter. For this purpose, the results obtained by using only high exposure images of our test data will be compared. To make the comparison fair, we use the same training database with our TLR to re-train the state of the art deep learning detector.

**Table 1** HDR: Confusion matrix without/with ROI; the results with the ROI are recorded in brackets [22]

	HARL	VARL	HAGL	VAGL	LVL	RVL	GAL	RAL	AI	GPL	RPL	OFRL
HARL	<b>441(441)</b>											
VARL		<b>783(783)</b>										
HAGL			<b>568(468)</b>									
VAGL				<b>549(549)</b>								
LVL					<b>1152(66)</b>	<b>72(0)</b>						
RVL						18(0)	<b>864(33)</b>					
GAL						6(6)	<b>180(180)</b>					
RAL								90(90)				
AI									<b>72 (72)</b>	<b>117 (117)</b>		
GPL												
RPL										<b>162 (162)</b>		
OFRL											<b>33 (39)</b>	

**Table 2** The precision and recall without/with ROI; the results with ROI are recorded in brackets [22]

	HARL	VARL	HAGL	VAGL	LVL	RVL	GAL	RAL	AL	GPL	RPL	OFRL	Average
Recall (%) without ROI/(with ROI)	100 (100)	97.4 (98.1)	98.1 (97.3)	97.3 (97.3)	98.5 (100)	92.3 (100)	100 (100)	100 (100)	100 (100)	92.9 (92.9)	100 (100)	100 (100)	98 (99)
Precision (%) without ROI/(with ROI)	100 (100)	100 (100)	96.8 (96.8)	92.8 (100)	98 (100)	96.8 (96.8)	100 (100)	100 (100)	100 (100)	95.1 (95.1)	98.2 (98.2)	91.7 (100)	97.5 (98.9)

**Table 3** HDR: Detection rate without/with ROI; the test results with ROI are recorded in brackets [22]

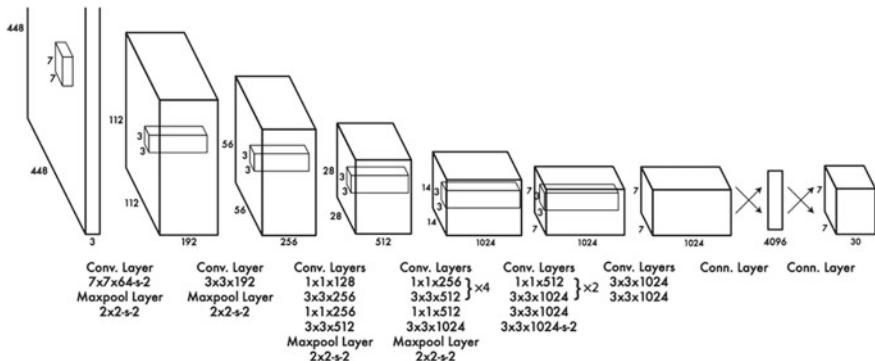
	HARL	VARL	HAGL	VAGL	LYL	RVL	GAL	RAL	AL	GPL	RPL	OFRL	Average
Ground truth without ROI/(with ROI)	447 (447)	804 (804)	477 (477)	567 (567)	1311 (66)	921 (36)	192 (192)	93 (93)	72 (72)	132 (132)	174 (174)	39 (39)	
Detection rate (%) without ROI/(with ROI)	98.7 (98.7)	974 (97.4)	98.1 (98.1)	100 (100)	94.7 (100)	95.8 (100)	96.9 (96.9)	96.8 (96.8)	100 (100)	93.2 (93.2)	94.8 (94.8)	92.3 (100)	96 (97.9)

For an autonomous vehicle application, object detector is selected based on two criteria: (1) real-time; (2) high accuracy. Although there have been a few deep learning object detectors, like Faster-RCNN [36], most of them cannot be run in real-time. In this chapter, we adopt You Only Look Once (YOLO) [37] as state of the art to be compared with our approach. YOLO can run in real-time and its new version (YOLOv2) [38] has achieved better performance than others, like Faster R-CNN [36] and single shot multibox detector (SSD) [39].

The architecture of YOLOv1 is shown in Fig. 8. It has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. The convolutional layers are pretrained on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection. The final output of YOLO is the  $7 \times 7 \times 30$  tensor of predictions. YOLO looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes YOLO is extremely fast, more than  $1000 \times$  faster than R-CNN and  $100 \times$  faster than Fast R-CNN.

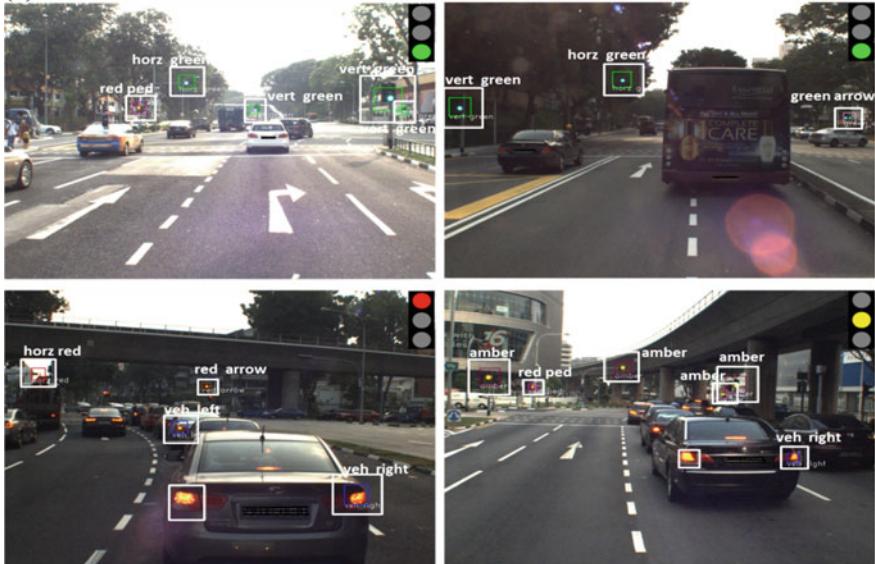
Some experimental results obtained by HDR and YOLOv2 are shown in Fig. 9a and b, respectively. The test results by using YOLOv2, similar to the last section for HDR, are shown in Tables 4, 5 and 6.

The true accuracies of the precision and recall are calculated by taking into account of the detection rate, i.e. multiplying the precision or recall rate with the detection rate, respectively. Table 7 lists the true precision and recall rate for YOLOv2 and our approach where the rates with ROI are recorded in brackets. Better performance can be achieved by our approach, either with or without ROI, compared with that of YOLOv2.

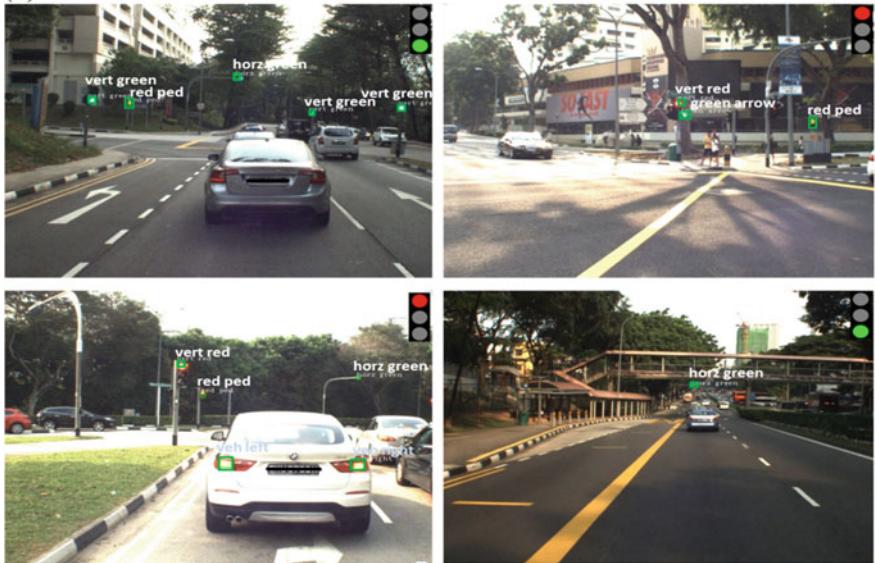


**Fig. 8** The Architecture of YOLO [37]. YOLO has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. The convolutional layers are pretrained on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection. The final output of YOLO is the  $7 \times 7 \times 30$  tensor of predictions

(a)



(b)



**Fig. 9** Some experimental results obtained by **a** HDR **b** YOLOv2 [22]

**Table 4** YOLOV2: Confusion matrix of the test results without/with ROI; the test results with the ROI are recorded in brackets [22]

**Table 5** YOLOv2: The precision and recall without/with ROI; the results with ROI are recorded in brackets [22]

	HARL	VARL	HAGL	VAGL	LVL	RVL	GAL	RAL	GPL	RPL	OFRL	Average
Recall (%) without ROI/(with ROI)	97.9 (97.9)	97.3 (97.3)	97.4 (97.4)	96.3 (96.3)	95.6 (95.5)	93.2 (100)	100 (100)	100 (100)	90.2 (90.2)	100 (100)	100 (100)	97.3 (97.9)
Precision (%) without ROI/(with ROI)	97.9 (97.9)	98.8 (98.8)	98.7 (98.7)	96.3 (96.3)	94.6 (95.5)	94.4 (100)	96.7 (96.7)	92.9 (92.9)	100 (100)	92.5 (92.5)	98.2 (98.2)	83.3 (84.6)

**Table 6** YOLOv2: Detection rate without/with ROI; the test results with ROI are recorded in brackets [22]

	HARL	VARL	HAGL	VAGL	LVL	RVL	GAL	RAL	AL	GPL	RPL	OFRL	Average
Ground truth without ROI/(with ROI)	447 (447)	804 (804)	477 (477)	567 (567)	1311 (66)	921 (36)	192 (192)	93 (93)	72 (72)	132 (132)	174 (174)	39 (39)	
Detection rate (%) without ROI/(with ROI)	96.6 (96.6)	97.0 (97.0)	96.9 (96.9)	99.5 (99.5)	93.6 (95.6)	99.3 (100)	95.3 (95.3)	90.3 (90.3)	95.8 (95.8)	90.9 (90.9)	94.8 (94.8)	92.3 (100)	95.2 (96.1)

**Table 7** Comparison of the precision and recall between HDR and YOLOv2; the results with ROI are recorded in brackets [22]

	YOLOv2	HDR
Recall (%)	92.6 (94.3)	94.1 ( <b>96.9</b> )
Precision (%)	90.8 (92.5)	93.6 ( <b>96.8</b> )

In details, with ROI, the improvement of the precision and recall are from 92.5% to 96.8% and from 94.3% to 96.9%, respectively.

The use of dark channel for detecting traffic light candidate is an efficient way to prevent many false positives caused by the traffic signs, sunlight, clothes of the pedestrian etc. This is because their corresponding regions on the dark image are not very visible. Figures 10, 11 and 12 give a few examples show that some false positives detected when only single color camera is used can be prevented by our dual channel approach. In Fig. 10, YOLOv2 results contain two false positives: the reflection of the traffic light on the bus body and sunlight on the building. Our dual channel approach prevents these two false positives successfully as no response for these two false positives can be found in the dark image in Fig. 10.



**Fig. 10** YOLOv2 versus HDR [22]. Left: YOLOv2, two false positives (in red circles) caused by the reflection of the traffic light on the bus body and the sunlight on the building, respectively; Middle: HDR, the two false positives in YOLOv2 are prevented because there is no response for these two false positives in the dark image (right)



**Fig. 11** YOLOv2 versus HDR [22]. Left: YOLOv2, one false positive (in red circle); Middle: HDR, the false positive in YOLOv2 is prevented in HDR because there is no response for this false positive in the dark image (right)



**Fig. 12** YOLOv2 (top) versus HDR (bottom) [22]. The false positives in the top caused by traffic sign and pedestrian can be prevented in the bottom

**Table 8** Comparison of computation cost (for one frame) between HDR and YOLOv2 [22]

	With ROI (ms)	Without ROI (ms)	Time saving
HDR	35	130	77%
YOLOv2	40	40	0

The results obtained on the same dataset have shown that the proposed approach in this chapter is better than the state of the art technique in terms of speed and accuracy.

The dual-channel algorithms presented in this paper are implemented in C++ on a Mini-PC (GIGABYTE, NVIDIA GeForce GTX 760), and can run in about 30–40 fps depending on the number of the traffic lights on an image. The processing time can be saved significantly by using ROI technology, see Table 8. On the contrary, YOLOv2 does not save time even uses the ROI because the network requires an image with fixed image size as input.

Our method has been demonstrated on real roads using A\*STAR IIR AV [40] via Data Distribution Service (DDS) [41]. One of the test results are shown in Fig. 13 where a few frames of a video at ten frame intervals are provided. For the interested readers, please refer to [22] or link for more video demo:

[https://www.youtube.com/watch?v=HQqaAvuJI\\_I](https://www.youtube.com/watch?v=HQqaAvuJI_I)

## 6 HDR Imaging Vehicle Signal Recognition

Vehicle following is one of the fundamental functions of an autonomous vehicle. It is important to detect and recognize tail light signals to prevent an autonomous vehicle from rear-end collisions or accidents. A cost-effective approach is expected although sensors like acoustic sonar or commercialized Advanced Driving Assistance System (ADAS) products such as mobileye could be used for rear-end collision warning.

**Fig. 13** The TLR results of a video [22]. The frames are shown at ten frame intervals, from left to right, top to bottom



Encouraged by the good performance in TLR, we have extended the dual-channel method presented in this chapter to Vehicle Signal Recognition (VSR) [20, 23]. The vehicle signal light, similar to traffic light, could be recognized robustly from dark channel where the lights are with clean background. Similar to TLR, our VSR is a two-stage approach: vehicles are detected from bright images using deep learning detector and then the signal light is then recognized from dark images using CNN. Unlike previous vehicle signal recognition approaches where pair taillight has to be extracted explicitly, we use vehicle appearance image instead.

Due to the length limitation, only Brake Light Recognition (BLR) is discussed in this chapter. Other signal lights, e.g. left turn or right turn, can be recognized in a way similar to the brake light recognition although video sequence analysis, e.g. LSTM [42], rather than single image is needed.

## 6.1 Related Approaches

Vehicle signal recognition has been explored by various approaches. The existing approaches can be roughly divided into two categories: temporal information [43–48] or single image [49–52]. Most of them detect signal lights using red color features and then try to pair taillights using their symmetry property. Cui et al. [49] proposed a hierarchical approach to detect vehicle and tail-lights in the daytime. They adopted Deformable Part Model (DPM) [53] to detect vehicle from images. The red light candidates are then found by clustering pixels in HSV color space within the bounding boxes of the candidates. After pairing taillight based on the prior knowledge about the vehicle appearance, a sparse dictionary is learned to classify signal lights. The approach is hard to be applied to autonomous vehicle because the slow processing speed, occlusion and possible false positives. Besides the slow detection problem for DPM, a serious problem of this approach is that the tail-lights could be occluded which resulting in taillight pairing consequently results in failure. In addition, the noise from the urban road environment, e.g. traffic lights, streetlight, could affect the detection of the tail-lights.

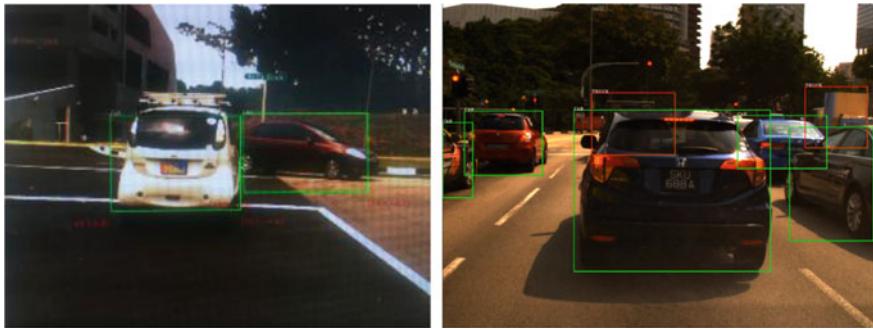
## 6.2 Two-Stage Vehicle Signal Recognition

The use of our dual-channel mechanism makes it possible to separate detection and recognition as two stages which can run in different exposure images. Similar to TLR mentioned above, we separate the VSR as two stages: vehicle detection and signal light recognition. The first stage will be executed with high exposure/bright image and the second stage will be run with low exposure/dark image.

## 6.3 Vehicle Detection

Although deep learning object detection has achieved very promising results, only a few of them can run in real-time. Based on very deep VGG-16 model [54], faster RCNN [36], one of the state of the art object detectors, can only reach frame rate at 5 frame per second, is far from the AV requirements (at least 10 frame per second is required because more perception module could share the computer source).

YOLO [37] and SSD [39] are examples of object detector can run in real-time. The detection is formulated as a regression problem in YOLO. As it access image only once, a fast frame rate is achieved. In this chapter, consistent with the comparison made in Sect. 5.1, YOLOv2 [38] is adopted as state of the art object detector to detect vehicle from a signal image. As we mentioned in Sect. 5.2, YOLOv2 [38] has achieved better performance than others, like Faster R-CNN [36] and SSD [39].



**Fig. 14** Vehicle detection with YOLOv2

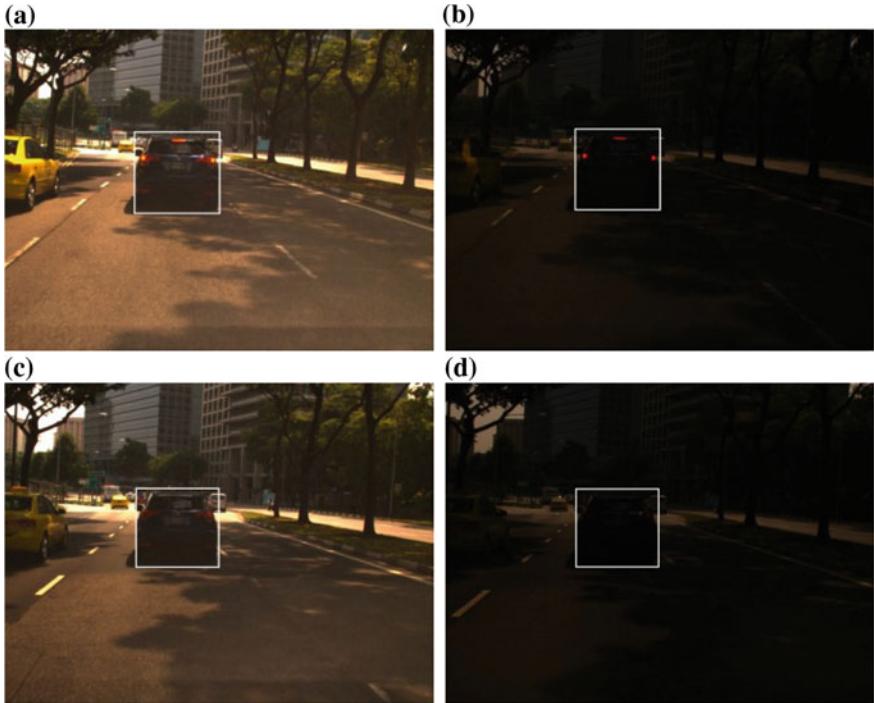
We retrain YOLOv2 detector with our database collected on real roads by using our autonomous vehicle. 21,000 images are annotated, results in more than 100,000 vehicle samples (about four to five vehicles in each image). We define eight classes of objects: (1) car; (2) truck; (3) lorry; (4) van; (5) bus; (6) motor cycles; (7) bicycle; (8) pedestrian. Two examples of the vehicle detection results are shown in Fig. 14.

#### 6.4 Brake Light Pattern and Recognition

Unlike existing BLR methods which require one to explicitly extract left and right tail-lights, appearance based deep learning is proposed in this chapter to recognize brake lights. In other words, the regions, we call it Brake-Light Pattern (BLP) in this chapter, within the bounding boxes detected by YOLOv2 detector are directly used to recognize brake light.

State of the art performance has been achieved by deep learning on a number of image recognition benchmark databases, e.g. ILSVRC-2012 [55]. Similar to the TLR presented in the previous sections, we state that the brake-light can be learned well from dark images than from bright images. Besides the clean background of dark images makes the lights recognition robust, the occlusion problem could be overcome to some extent by using BLPs, see Fig. 15, proposed in this chapter rather than a pair of tail-lights. Furthermore, the middle brake-light included in the BLP, most of them are located at the rear window of vehicles, makes the recognition more reliable than that using only left and right tail-lights. The previous approaches do not use this middle light because it is hard to extract this relatively darker light compared with the left and right tail lights.

An example is shown in Fig. 15. The BLPs of a vehicle, corresponding to their bounding boxes in the left (bright images), are shown in the right (dark images). The brake light can be recognized accurately from dark images because the difference between the “braking” and “normal” on a dark image is much large than that on their counterpart on bright image.



**Fig. 15** The Brake-Light Pattern (BLP) of a vehicle (right, dark image) corresponding to the vehicle detection bounding box shown in the left (bright image)

## 6.5 Experimental Results

Similar to TLR, no public benchmark is available for brake-light recognition, especially HDR benchmark. Most of the brake light recognition systems use bright image only. Nevertheless, in this chapter, the quantitative analysis of our proposed dual-channel method has been done based on our own database. The comparison with the approach which use only bright image is provided to show the advantages of our approach.

The same videos used in the TLR, see Sect. 3, have been used to train and evaluate brake light recognition. As vehicle detection results are the same for previous approach and our approach (same bright image and same detector), the brake light recognition results, using bright image in previous approaches and dark image in our approach, are compared.

The ground truth (“normal” or “braking”) comes from 1,001 images containing 2,123 samples. In order to train network, we generate about one million samples from above seed samples using the same way described in Sect. 5.1. Some training samples are shown in Fig. 16. The images with “normal” and “braking” patterns are shown in the top and bottom rows, respectively.



**Fig. 16** Some training samples generated from seed images for “normal” (top row) and “braking” (bottom row)

The ten-fold evaluation is adopted to test the accuracy. The results are listed in Table 9.

The average accuracy of our method is found to be 97.5%, much better than that of previous approaches, 89%, obtained by using bright image. The vehicle detection rate is found to be 99.5%.

Figures 17 and 18 are two examples for brake light recognition experiments. The bounding box of vehicle is marked in green or red when it is identified as “normal” or “braking”, respectively. The method can solve partial occlusion problem (Fig. 18) because a pattern rather than pair light is used.

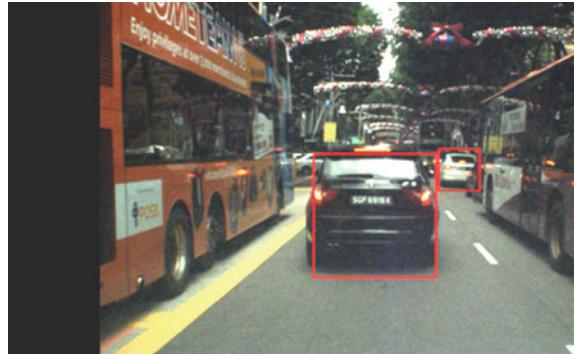
**Table 9** Comparison of the previous approaches (bright image) and our approach (dual channel)

	Previous approaches (bright image only)	Our approach (dual channel)
Accuracy (%)	89	97.5



**Fig. 17** Brake light recognition results [20]. Left: “normal” (green); right: “braking” (red)

**Fig. 18** Brake light recognition under partial occlusion conditions [20]. The brake status of the vehicle on the right can be recognized even its right tail light is fully occluded



Similar to the TLR presented in Sect. 5, the algorithms developed for VSR in this chapter have been integrated into our autonomous vehicle, A\*STAR IIR AV [40]. The demonstrations on real roads, including vehicle following, obstacle avoidance, etc., have shown that both the accuracy and the speed are satisfied with the autonomous vehicle requirement. Run the VSR and TLR together in the same PC presented in Sect. 5.2, i.e. Mini-PC (GIGABYTE, 2.5 Ghz CPU, GTX 760), we achieve 25–35 fps depending on the number of the traffic lights and vehicle signal lights on an image.

## 7 Conclusion and Future Work

A real-time TLR system has been proposed in this chapter to detect and recognize TL based on high dynamic range imaging and deep learning. The advantages of a HDR camera, i.e. multiple exposure images, are fully used. The drawback of the state of the art, which uses only bright images and false positives could be caused, can be overcome by our approach because the low exposure image has clean background (dark) that ensures the TL can be detected reliably. Furthermore, the candidates on the high exposure image, corresponding to the one on the dark image, can be recognized with high accuracy because of rich context is available. The number of the TL candidates to be identified by CNN is significantly reduced by using saliency map and ROI. This makes it fast as well robust to noise, e.g. vehicles' tail lights. Finally, the accuracy and reliability are furtherly improved by developing a tracking technology. By executing the method on a large database collected from real roads, we have shown that the performance of our method is better than the state of the art. Encouraged by the good performance of the TLR, we extend our dual-channel method to VSR. Vehicles are detected from bright images and the vehicle signal lights are recognized from the counterpart dark images. Similar to TLR, good VSR performance has been achieved. The online tests on our autonomous vehicle have done successfully. It has been verified that our method satisfies the speed and accuracy requirements of an autonomous vehicle.

The investigations on using both dark and bright images as input to the CNN network could be done in the near future. The quantitative performance at night could be done. Actually, during the test on real road, we have observed that our dual-channel method is feasible at night. This is because that the night effects are not be high when we detect traffic lights from dark images. Proper camera parameters and re-trained CNN model with night data would be sufficient for night performance. What we need to do is nothing but adjusting camera parameters properly and re-training the CNN with night data. Lastly, the RNDF (Route Network Definition File) could be adopted in the future to locate traffic lights. It is clear that the false positives can be eliminated significantly by fusing with RNDF information.

**Acknowledgements** We have benefited enormously from ideas and discussions with our ex-colleagues: Yu Pan, Serin Lee, Zhi-Wei Song, Boon-Siew Han and Vincensius-Billy Saputra.

## References

1. Jensen, M.B., Philipsen, M.P., Trivedi, M., Mogelmose, A., Moeslund, T.: Vision for looking at traffic lights: issues, survey, and perspectives. *IEEE Trans. Intell. Transp. Syst.* **17**(7), 1800–1815 (2016)
2. Diaz, M., Pirlo, G., Ferrer, M.A., Impedvov, D.: A survey on traffic light detection. In: Proceedings of ICIAP 2015 workshops on New Trends in Image Analysis and Processing, Lecture Notes in Computer Science, vol. 9281, pp. 201–208 (2015)
3. Philipsen, M.P., Jensen, M.B., Mogelmose, T., Moeslund, T.B., Trivedi, M.M.: Ongoing work on traffic lights: detection and evaluation. In: Proceedings of 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) (2015)
4. Gong, J., Jiang, Y., Xiong, G., Guan, C., Tao, G., Chen, H.: The recognition and tracking of traffic lights based on color segmentation and CAMSHIFT for intelligent vehicles. In: Proceedings of IEEE Intelligent Vehicle Symposium (2010)
5. Siogkas, G., Skodras, E., Dermatas, E.: Traffic lights detection in adverse conditions using color, symmetry and spatiotemporal information. In: Proceedings of International Conference on Computer Vision Theory and Applications, pp. 620–627 (2012)
6. Charette, R., Nashashibi, F.: Traffic light recognition using image processing compared to learning processes. In: Proceedings of IEEE/RSJ International Conference on Robots and Systems, pp. 333–338 (2009)
7. Diaz-Cabrera, M., Cerri, P., Sanchez-Medina, J.: Suspended traffic lights detection and distance estimation using color features. In: Proceedings IEEE International Conference on Intelligent Transportation Systems, pp. 1315–1320 (2012)
8. Levinson, J., Askeland, J., Dolson, J., Thrun, S.: Traffic light mapping, localization, and state detection for autonomous vehicles. In: Proceedings of International IEEE Conference on Robotics and Automation (ICRA), pp. 5784–5791 (2011)
9. Haltakov, V., Mayr, J., Unger, C., Ilic, S.: Semantic segmentation based traffic light detection at day and at night. In: Proceedings of German Conference on Pattern Recognition, Lecture Notes in Computer Science, vol. 9358, pp. 446–457 (2015)
10. Charette, R., Fawzi Nashashibi, F.: Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates. In: Proceedings of IEEE Intelligent Vehicles Symposium (2009)
11. Fairfield, N., Urmsom, C.: Traffic light mapping and detection. In: Proceedings of International IEEE Conference on Robotics and Automation (ICRA), pp. 5421–5426 (2011)

12. John, V., Yoneda, K., Qi, B., Liu, Z. Mita, S.: Traffic light recognition in varying illumination using deep learning and saliency map. In: Proceedings of International IEEE Conference on Intelligent Transportation System (ITSC) (2014)
13. Gradinescu, V., Gorgorin, C., Diaconescu, R., Cristea, V., Iftode, L.: Adaptive traffic lights using car-to-car communication. In: Proceedings of 65th IEEE Vehicular Technology Conference, pp. 21–25 (2007)
14. Kumar, N., Lourenco, N., Terra, D., Alves, L.N., Aguiar, R.L.: Visible light communication in intelligent transportation systems. In: Proceedings of IEEE Intelligent Vehicle Symposium, pp. 748–753 (2012)
15. Dresner, K., Stone, P.: A multiagent approach to autonomous intersection management. *Artif. Intell. Res.* **31**, 591–656 (2008)
16. High Dynamic Range. [https://en.wikipedia.org/wiki/High-dynamic-range\\_imaging](https://en.wikipedia.org/wiki/High-dynamic-range_imaging)
17. Jang, C., Kim, C., Kim, D., Lee, M., Sunwoo, M.: Multiple exposure images based traffic light recognition. In: Proceedings of IEEE Intelligent Vehicle Symposium, pp. 1313–1318 (2014)
18. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: Proceedings of International IEEE Conference on Computer Vision and Pattern Recognition, pp. 886–893 (2005)
19. Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., Wu, Y.: Learning fine-grained image similarity with deep ranking. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1386–1393 (2014)
20. Wang, J.-G., Zhou, L.-B., Pan, Y., Lee, S., Han, B.-S., Billy, V.: Appearance based brake-lights recognition. In: Proceedings of IEEE Intelligent Vehicle Symposium (2016)
21. Casares, M., Almagambetov, A., Velipasalar, S.: A robust algorithm for the detection of vehicle turn signals and brake lights. In: Proceedings of International IEEE Conference on Advanced Video and Signal-Based Surveillance, pp. 386–391 (2012)
22. Wang, J.-G., Zhou, L.-B.: Traffic light recognition with high dynamic range imaging and deep learning. *IEEE Trans. Intell. Transp. Syst.* **20**(4), 1341–1352 (2019)
23. Wang, J.-G., Zhou, L.-B., Song, Z.-W., Yuan, M.-L.: Real-time vehicle signal lights recognition with HDR camera. In: Proceedings of IEEE International Conference on Internet of Things (iThings) (2016)
24. Saliency Map. [https://en.wikipedia.org/wiki/Saliency\\_map](https://en.wikipedia.org/wiki/Saliency_map)
25. Kim, H.-K., Park, J.H., June, H.-Y.: Effective traffic lights recognition method for real time driving assistance system in the daytime. *Int. J. Electr. Comput. Eng.* **5**(11), 1429–1432 (2011)
26. Bradski, D.: Dr. Dobb's journal of software tools
27. Zebra2 camera. <https://www.ptgrey.com/zebra2-28-mp-color-gige-hd-sdi-sony-icx687-camera>
28. Lu, H., Zhang, H., Yang, S., Zheng, Z.: Camera parameters auto-adjusting technique for robust robot vision. In: Proceedings of International IEEE Conference on Robotics and Automation, pp. 1518–1523 (2010)
29. Agarwal, V., Abidi, B.R., Koschan, A., Abidi, M.A.: An overview of color constancy algorithms. *J. Pattern Recognit. Res.* **1**(1), 42–54 (2006)
30. Shim, I., Lee, J.-Y., Kweon, I.S.: Auto-adjusting camera exposure for outdoor robotics using gradient information. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robotics and Systems, pp. 1011–1017 (2014)
31. Deep learning. Wiki. [https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning)
32. Hu, Y., Xie, X., Ma, W.-Y., Chia, L.-T., Rajan, D.: Salient region detection using weighted feature maps based on the human visual attention model. In: Proceedings of Pacific Rim Conference on Multimedia (2004)
33. Achanta, R., Hemami, S., Estrada, F., Susstrunk, S.: Frequency-tuned salient region detection. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2009)
34. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of NIPS (2012)
35. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of ACM International Conference on Multimedia, pp. 675–678 (2014)

36. Ren, S., He, K., Girshic, R., Sun, J.: Faster R-CNN: toward real-time object detection with region proposal networks. <https://arxiv.org/abs/1506.01497>
37. YOLO: real-time object detection. <https://pjreddie.com/darknet/yolov1/>
38. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. <https://arxiv.org/abs/1612.08242>
39. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C.: SSD: single shot multibox detector. <https://arxiv.org/abs/1512.02325>
40. IIRAV. <https://www.a-star.edu.sg/i2r/RESEARCH/AUTONOMOUS-SYSTEMS>
41. Data Distribute Service. [https://en.wikipedia.org/wiki/Data\\_Distribution\\_Service](https://en.wikipedia.org/wiki/Data_Distribution_Service)
42. Long short-term memory. [https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)
43. Koller, D., Weber, J., Malik, J.: Robust multiple car tracking with occlusion reasoning. Springer, Berlin (1994)
44. She, K., Bebis, G., Gu, H., Miller, R.: Vehicle tracking using online fusion of color and shape features. In: Proceedings of 7th IEEE International IEEE Conference on Intelligent Transportation Systems, pp. 731–736 (2004)
45. Chan, Y.-M., Huang, S.-S., Fu, L.-C., Hsiao, P.-Y.: Vehicle detection under various lighting conditions by incorporating particle filter. In: Proceedings of IEEE Intelligent Transportation Systems Conference, pp. 534–539 (2007)
46. Malley, R., Jones, E., Glavin, M.: Rear-lamp vehicle detection and tracking in low-exposure color video for night conditions. IEEE Trans. Intell. Transp. Syst. **11**(2), 453–462 (2010)
47. Casares, M., Almagambetov, A., Velipasalar, S.: A robust algorithm for the detection of vehicle turn signals and brake lights. In: Proceedings of IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance, pp. 386–391 (2012)
48. Almagambetov, A., Casares, M., Velipasalar, S.: Autonomous tracking of vehicle rear lights and detection of brakes and turn signals. In: Proceedings of IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA), pp. 1–7 (2012)
49. Cui, Z.-Y., Yang, S.-W., Tsai, H.-M.: A vision-based hierarchical framework for autonomous front-vehicle taillights detection and signal recognition. In: Proceedings of IEEE 18th International Conference on Intelligent Transportation Systems, pp. 931–937 (2015)
50. Thammakaroon, P., Tangamchit, P.: Predictive brake warning at night using taillight characteristic. In: Proceedings of IEEE International Symposium on Industrial Electronics, pp. 217–221 (2009)
51. Ming, Q., Jo, K.-H.: Vehicle detection using tail light segmentation. In: Proceedings of 6th IEEE International Forum on Strategic Technology (IFOST), vol. 2, pp. 729–732 (2011)
52. Nagumo, S., Hasegawa, H., Okamoto, N.: Extraction of forward vehicles by front-mounted camera using brightness information. In: Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, vol. 2, pp. 1243–1246 (2003)
53. Felzenszwalb, P.F., Girshick, R.B., McAllester, D.: Cascade object detection with deformable part models. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 2241–2248 (2010)
54. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)
55. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein M.: Imagenet large scale visual recognition challenge. [arXiv:1409.0575](https://arxiv.org/abs/1409.0575) (2014)

# The Application of Deep Learning in Marine Sciences



Miguel Martin-Abadal, Ana Ruiz-Frau, Hilmar Hinz  
and Yolanda Gonzalez-Cid

**Abstract** Ecological studies are increasingly using video image data to study the distribution and behaviour of organisms. Particularly in marine sciences cameras are utilised to access underwater environments. Up till now image data has been processed by human observers which is costly and often represents repetitive mundane work. Deep learning techniques that can automatically classify objects can increase the speed and the amounts of data that can be processed. This ultimately will make image processing in ecological studies more cost effective, allowing studies to invest in larger, more robust sampling designs. As such, deep learning will be a game changer for ecological research helping to improve the quality and quantity of the data that can be collected. Within this chapter we introduce two case studies to demonstrate the application of deep learning techniques in marine ecological studies. The first example demonstrates the use of deep learning in the detection and classification of an important underwater ecosystem in the Mediterranean (*Posidonia oceanica* seagrass meadows), the other showcases the automatic identification of several jellyfish species in coastal areas. Both applications showed high levels of accuracy in the detection and identification of the study organisms, which represents encouraging results for the applicability of these methodologies in marine ecological studies. Despite its potential, deep learning has yet not been widely adopted in ecological studies. Information technologists and natural scientists alike need to more actively collaborate to move forward in this field of science. Cost-effective data

---

M. Martin-Abadal (✉) · Y. Gonzalez-Cid

Departament de Ciències Matemàtiques i Informàtica, Systems Robotics and Vision Group,  
Universitat de les Illes Balears, Ctra. Valldemossa Km 7.5, 07122 Palma, Spain  
e-mail: [miguel.martin@uib.es](mailto:miguel.martin@uib.es)

Y. Gonzalez-Cid

e-mail: [yolanda.gonzalez@uib.es](mailto:yolanda.gonzalez@uib.es)

A. Ruiz-Frau · H. Hinz

Department of Marine Ecosystem Dynamics, Esporles (Illes Balears), IMEDEA (CSIC-UIB),  
Institut Mediterrani d'Estudis Avançats, Miquel Marquès 21, 07190 Esporles, Spain  
e-mail: [anaruiz@imedea.uib-csic.es](mailto:anaruiz@imedea.uib-csic.es)

H. Hinz

e-mail: [hhinz@imedea.uib-csic.es](mailto:hhinz@imedea.uib-csic.es)

collection solutions are desperately needed in a time when large amounts of data are required to detect and adapt to global environmental change.

**Keywords** Deep learning · Application · Marine · *Posidonia oceanica* · Jellyfish · Semantic segmentation · Object detection

## 1 Introduction

Traditional data collection in ecological studies generally relies on human visual observations detecting the occurrences of organisms and relating those to environmental or anthropogenic factors. Similarly, visual observations are crucial in describing behavioural interactions amongst individuals of the same species and other organisms. Making such ecological observations is often time-consuming, labour intensive and hence costly [1–3].

The high associated cost of undertaking ecological observations often restricts the amounts of data that can be collected, therefore limiting the robustness of studies and the conclusions that can be drawn from the results. With the advent of relatively cheap video recording techniques, visual observations can now be made simultaneously at multiple sites covering larger spatial and temporal scales reaching environments where previously no human observations could be made. This is of particular relevance for ecological studies investigating organisms inhabiting underwater marine ecosystems. Here, human based observations are limited by constraints of depth and time. Human based observations through divers are generally limited to depths of approximately 30 m and may last only a couple of hours (depending on depth), while deeper depth can only safely be reached with increasingly more complex technologies [4–6]. Video cameras in contrast can easily be deployed to almost any depth and from any type of platforms (e.g. [7–11]). Video observations have thus increased dramatically the potential for data collection in marine sciences.

Nevertheless, these advances have not yet led to a reduction in the cost of ecological studies using images as a data source. While more data can now be collected, its subsequent interpretation and analysis is often still done by humans. This process is highly repetitive and often takes the same amount of time or longer than the recording of the original images thus keeping costs elevated [12, 13].

Computer aided automatic classification of images using deep learning can significantly increase the speed and thus the cost of image data interpretation and analysis. While expert knowledge is still needed to train and quality check the computer aided image interpretation, the automatising allows for the processing of larger data-sets within a fraction of the time a human observer would require. Additionally, the computer aided interpretations often have a higher precision compared to humans [14].

While in recent years there has been an increasing interest in the use of automatic image classification in ecology, there are currently still few scientists adopting these technological advances, probably due to the interdisciplinary know-how boundary

between ecology and new information technologies (but see e.g. [15–19]). This lack of uptake may however be overcome in the future as end-user-based interfaces for this technology become more user friendly.

Automated image classifications and segmentation through deep learning techniques represent a game changer for ecological studies using image-based data collection. It opens the possibility to increase data collection and processing to a completely new level with the potential of delivering more robust and statistically sound data at a highly reduced cost. The reduced cost may also provide the solution for the maintenance or establishment of highly important long-term data collection against the backdrop of anthropogenic change [20]. The collection of long-term data at the appropriate spatial and temporal scale has thus far lacked commitment by governments and scientists alike due to their high cost and initially low scientific returns respectively [21].

In this chapter we present two case studies that use deep learning to automatically process underwater images with the aim of showcasing the potential of these methodologies in improving ecological data collection and processing. The studies presented represent promising solutions for the data collection and processing of two marine organisms highly relevant for society.

The first case study demonstrates the identification of seagrass meadows, *Posidonia oceanica*, from video sequences recorded from an Autonomous Underwater Vehicle (AUV) using semantic segmentation. Seagrass meadows provide a wide range of benefits for society such as the attenuation of wave energy thus contributing to the maintenance of sandy beaches as well as providing a habitat for many commercial and non-commercial species. With the help of deep learning, larger and more precise habitat maps can be produced for long term monitoring, vital for the management and protection of this habitat.

The second case study shows how different species of jellyfish, some of them with negative impacts for society, can be identified and classified using object detection deep learning algorithms. This detection and assessment of jellyfish has relevance with respect to increasing our understanding of jellyfish ecology and also provides the potential for coastal monitoring systems to mitigate impacts of jellyfish on humans.

## 2 Methodology

Deep learning enables computational models composed of multiple processing layers to learn representations of data with different levels of abstraction. Deep Learning is one of the sub-fields of Machine Learning and has been advancing at an impressive pace over the last couple of years, bringing excellent results in different disciplines. In particular, Convolutional Neural Networks (CNN or ConvNet) [22] are achieving important milestones in image, video and audio processing and have been widely adopted by the computer vision community. A CNN is a particular kind of deep neural network consisting of an input layer, an output layer and multiple hidden

layers. The hidden layers of a CNN consist of diverse convolutional layers, RELU activation layers, pooling layers, fully connected layers and normalisation layers.

The wide range of algorithms and applications of CNN in computer vision can be classified into four main different types:

- Classification. Given a raw image the task is to identify the class which the image belongs to.
- Classification and Localisation. Given a raw image, with only one object in it, the task is to find the location of the object within the image.
- Object Detection. The task is to identify the location of several objects within an image. Objects might be of the same class or different classes altogether.
- Image Segmentation. Each pixel composing an image is classified and assigned to a particular class. Image segmentation is also known as semantic segmentation.

The main methodology and general requirements needed when implementing CNN for image processing purposes varies whether we are using it for classification, object detection, or segmentation.

Due to the vast resources required to train deep learning architectures or the large and challenging data-sets on which deep learning models should be trained, it is very common to use *transfer learning* instead of designing a model from scratch. In transfer learning, a model trained in order to perform one task is re-trained to accomplish a second related task, which allows an improved performance when modelling the second task.

Therefore, the first step using transfer learning is to select a pre-trained source model from already available models that best fits the application needs. If the dataset in your problem domain is similar to ImageNet dataset [23], a pre-trained model on this dataset can be used. The most widely used pre-trained models are VGG net [24] with 19 or 16 layers, ResNet [25] with 152, 101, 50 layers or less, DenseNet [26] with 201, 169 and 121 layers, Inception [27] or Xception [28].

The next step is to organise the data needed to train the selected model. Data should be divided in two subsets, the training set and the testing set. A ground truth (GT) for both subsets should also be generated. GT images are those labelled by experts using direct observation, from which the network will learn during the training.

Training deep neural networks is difficult. It requires knowledge and expertise in order to properly train and obtain an optimal model. Different model training algorithms may require different hyperparameter tuning. The hyperparameters change some network's features or its training process, and are fixed before the training process begins.

In general, the values of the hyperparameters are chosen by training the network several times with different values and deciding which ones work best by evaluating the results.

Moreover, during the training process there are different methods that can be used to improve the validation. One of the most commonly used is cross validation. For each combination of hyperparameters the model is trained using the *K*-fold Cross Validation method [29], splitting the data into *X* subsets of the same size and

training the network  $X$  times, each time making use of one subset to test the network and the remaining  $X - 1$  subsets to train it. This method achieves a reduction of the results variability, obtaining a more accurate performance estimation in the validation process.

From each cross-validation training applied over a set  $H$  of hyperparameters,  $X$  models are generated,  $M_H^i$  where  $H = 1, 2, \dots, h$  represents the hyperparameter set number and  $i = 1, 2, \dots, X$  the model index. Subsequently, the  $X$  models are executed over their corresponding test subset, obtaining the predictions,  $P_H^i$ . From these predictions, each model is evaluated, assessing its performance,  $R_H^i$ . Note that depending on the model trained and its output (classification, localisation, object detection or image segmentation) the metrics to be used and the evaluation process might be different. Finally, the performance  $R_H$  of each set  $H$  of hyperparameters is obtained by computing the mean of its  $X$  models performance  $R_H^i$ .

The workflow for assessing the performance of each hyperparameter set is represented in Fig. 1.

The next sections show the training and validation process of two different deep ConvNet for automatically *Posidonia oceanica* segmentation and jellyfish detection and classification, respectively.

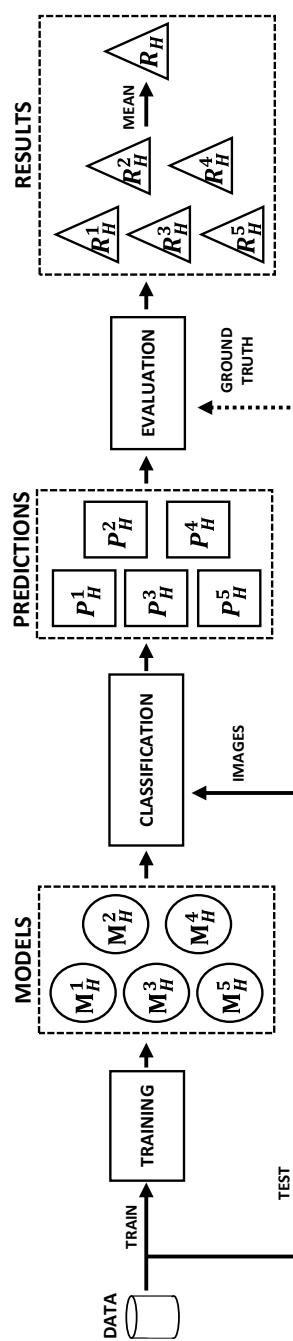
### 3 Seagrass Segmentation

*Posidonia oceanica* is an endemic Mediterranean seagrass species that forms dense and extensive meadows that grow down to a depth of 45 m. From a social-ecological perspective, this ecosystem is of up-most importance, since it plays a crucial role in the maintenance of coastal processes and functions and provides a range of benefits and services to society [30, 31]. Recent studies have evidenced a globally decline of *P. oceanica* [32, 33]. As a result of the previous statements, the European Commission directive 92/43/CEE identifies *P. oceanica* as a priority natural habitat.

The management and restoration strategies for *P. oceanica* heavily rely on aspects such as the monitoring and mapping of the coverage and state of the meadows. These aspects are fundamental in the assessment of *P. oceanica* conservation status, allowing to prematurely detect decline trends, or assess how effective an applied protection and recovery measure is.

Currently, the monitoring tasks are mostly carried out by divers, measuring in a manual manner meadow parameters such as lower limit depth, shoot density or extension [34]. However, the collection of these data is slow and costly.

Other monitoring approaches *P. oceanica* make use of multi-spectral satellite images [35], acoustic bathymetries [36] or Autonomous Underwater Vehicles (AUV) equipped with sensors to obtain different parameters from *P. oceanica* meadows [37, 38]. These techniques suffer from some disadvantages, some of them are the poor effectiveness in large depth areas, the inability to distinguish between *P. oceanica* and other algae types or the fact that they can not perform the detection autonomously.



**Fig. 1** Hyperparameter set “H” workflow. The network is trained X times making use of the training dataset and the k-fold cross validation method, outputting X models (here X=5). Following, the models are evaluated over the test dataset. Finally, the evaluation metrics are calculated from the X models mean performance

In [39] an autonomous detection was achieved by combining traditional image descriptors with *Machine Learning* (ML) and the use of *Support Vector Machines* (SVM). Additionally, in [40] the idea of using *Convolutional Neural Networks* (CNN) for *P. oceanica* detection is explored. These approaches also have some inconveniences, in both of them the classification is not made at a pixel level, instead, the images are divided into smaller patches, classifying each patch as as *P. oceanica* or as background. This causes loss of information and lower prediction resolution since all pixels of a patch are imposed the classification class of the patch they belong to.

The application of deep learning techniques allows for the use of neural network architectures with more hidden layers that, along with a semantic segmentation classification, can perform a per pixel classification instead of a patch-based one, avoiding the information loss and obtaining a full-image resolution classification, obtaining an improved *accuracy* in the classification task.

The main goal in this case study was to perform an automatic segmentation of *P. oceanica* meadows in sea-floor images.

The following sections describe the deep neural network used and its main characteristics, expose the different study cases and hyperparameter combinations, the data acquisition and processing, the validation and evaluation processes, and finally, the classification results.

### 3.1 Deep Learning Approach

To determine the areas where *P. oceanica* was present, a semantic segmentation architecture was used. Subsequently, we describe the architecture of the network used and its training details.

#### Network Architecture

The architecture used is the so called VGG16-FCN8, which is a fully convolutional network, meaning that can make dense pixel-wise predictions for image tasks like semantic segmentation. These architectures are divided into two blocks, the encoder and the decoder.

The encoder extracts spatial features from the input images making use of a series of convolutional layers. These layers apply a convolution by sweeping a kernel over the input and passing the result to the following layer. Time this process is carried out X times over the same input but using a different kernel, generating X feature maps. Also, encoders implement max pooling layers, which used to reduce the feature maps dimension, offering a better computational performance as the number of parameters is reduced.

The selected architecture makes use of the VGG16 encoder [24], subtracting the last classification layer and converting the last two fully connected layers into convolutional layers. It contains six different sections, the first five of them are constructed by two or three convolutional layers and a max pooling layer. The last one, contains two convolutional and two drop out layers interleaved. This structure

allows to extract low-level coarse information from the image on the first sections, and then, as more convolutional and max pooling layers are applied, the feature maps shrink up to a 1/32 of the original image size, incorporating more complex high-level information. Finally, the convolutional layers of the last section maintain the spatial information into the decoder and generate a low resolution segmentation while the drop out layers help to reduce overfitting.

The decoder purpose is to take the low resolution segmentation output of the encoder and up-sample it to the original image size, obtaining a high-resolution segmentation of it. In order to accomplish this task, a series of transposed convolutional layers are used. These layers apply an inverse convolution over the input, up-sampling each pixel to the convolutional kernel size. The decoder also contains skip layers [41], which are used to integrate the encoder's low level features to higher level, coarse information from the transposed convolutional layers. Lastly, an activation layer obtains the final semantic segmentation.

The selected architecture makes use of the FCN8 decoder [42], which contains three of the aforementioned transposed convolutional layers and three skip layers interleaved. By adjusting the kernel sizes and strides of the transposed convolutional layers, the shrunked feature maps are up-sampled into the original image size. Lastly, a softmax activation layer obtains the final probabilistic segmentation map. The explained architecture is presented in Fig. 2.

This architecture has already been used for other segmentation tasks, like road segmentation for autonomous drive in [43] or class segmentation of the PASCAL VOC 2011-2 dataset in [42]. Always presenting great results

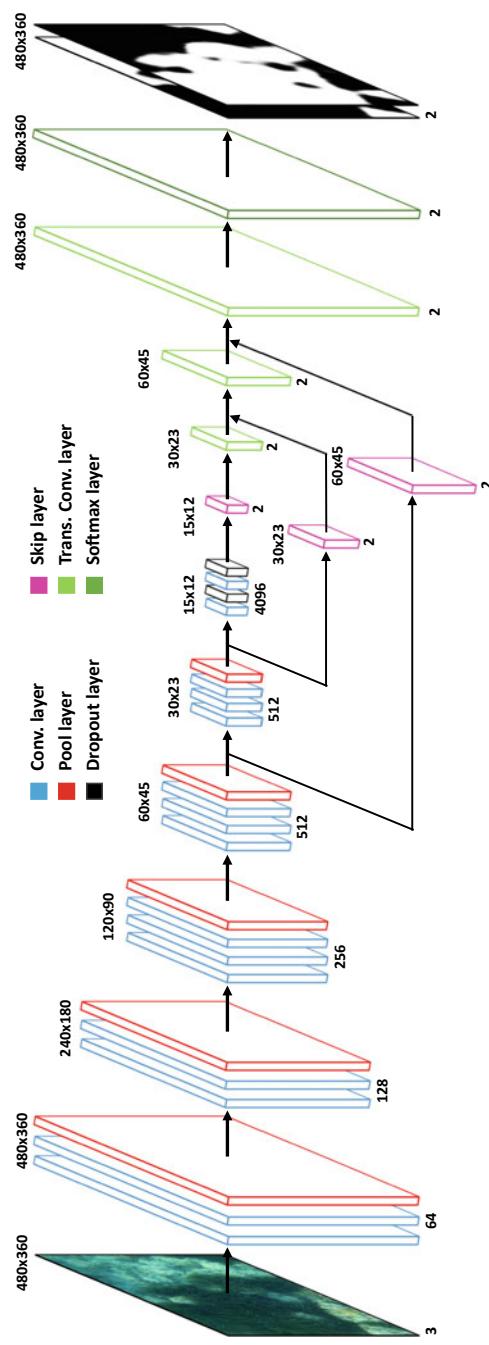
## Training Details

In order to train the VGG16-FCN8 architecture, both encoder and decoder should be trained. Their training is conducted by means of readjusting the kernel values in the convolutional layers and transposed convolutional layers, respectively. This architecture allows to train both encoder and decoder with the same back propagation functions, allowing its training in a single forward and backward pass for each iteration.

The training process makes use of images containing *P. oceanica*, and their corresponding label maps, where each class is marked in a different colour.

To train the network a backpropagation function is needed, indicating the direction and magnitude of change. In this case, a cross-entropy loss function is used [44], its loss increases as the predicted probability diverges from the ground truth label. Also, the Adam optimization algorithm is implemented in order to help the training reach the global minimum error [45]. Finally, in order to help preventing overfitting [46], two dropout layers are interleaved between the fully connected layers of the encoder.

In order to benefit from the advantages of transfer learning, the encoder layers are initialized with the pretrained weights of a VGG network trained on ImageNet [47]. The initialization of the transposed convolution layers of the decoder is carried out using bilinear upsampling. Finally, a truncated Gaussian initialization is applied to the skip connections. These initialization parameters for this network have already presented great results in [43].



**Fig. 2** VGG16-FCN8 Neural network architecture. The encoder is conformed by convolutional layers (blue), pooling layers (red) and dropout layers (black). The decoder is conformed by skip layers (purple), transposed convolutional layers (green) and softmax layers (orange). For each layer it is indicated the number of feature maps (below) and their shape (above)

The trainings were performed on a computer equipped with an Intel Core i7-7700 processor, a GeForce GTX 1080 graphic card and 16 MB of RAM.

### **3.2 Experimental Framework**

This section describes the experimental framework followed in this application. First, the image acquisition and labelling processes are described, alongside with the dataset usage. Next, the different hyperparameter combinations studied are presented. Finally, we describe the validation and evaluation processes.

#### **Datasets**

##### *Acquisition*

The images used to train and test the architecture were extracted from video sequences recorded using cameras mounted on an AUV facing downwards.

An AUV was navigated over *P. oceanica* beds located on the West and North-West of Mallorca (Fig. 3), obtaining images under different *P. oceanica* conditions such as health state, meadow density and coloration; or water depth, illumination and turbidity.

A sample of the gathered images can be seen in Fig. 4.

##### *Labelling*

From the obtained images, label maps were manually built. The areas with *P. oceanica* were marked in white, and the background areas in black. These labels maps are used as ground for the gathered images, and are used to train and test the network. Figure 5 shows an image with its corresponding label map.

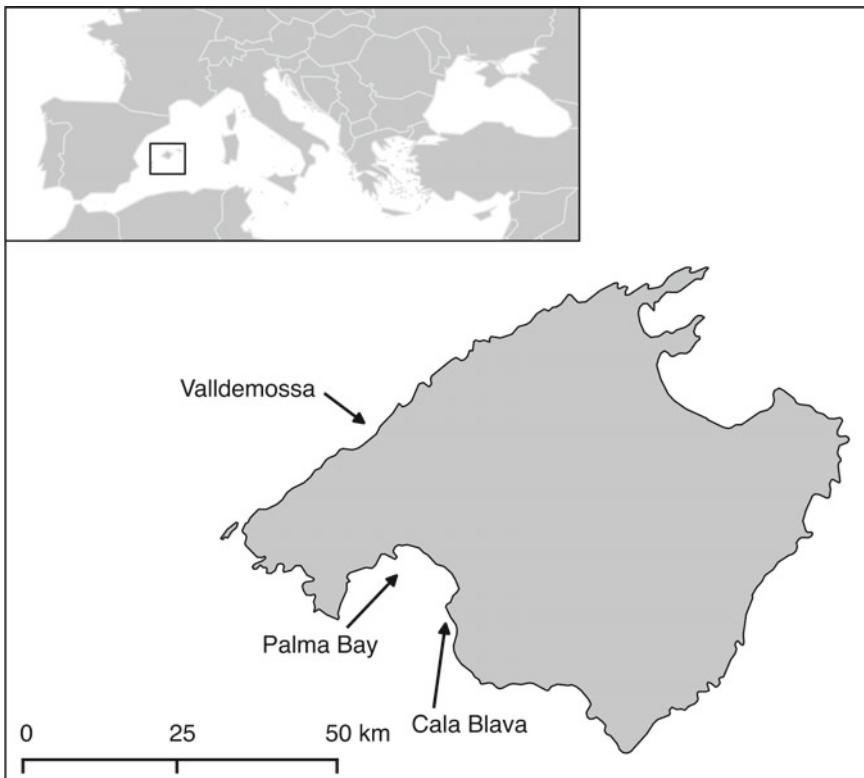
##### *Dataset Arrangement*

In order to build our datasets, six different different AUV missions were performed, obtaining up to 483 images. These images were representative of the different environmental and meadow conditions encountered during the sampling process.

From the gathered images, we generated two datasets, namely the *mix* dataset, including 460 images and the *extra* dataset containing 23 images. Table 1 indicates the location, month of acquisition, camera used, number of images and the corresponding dataset of each mission.

The *mix* dataset (460 images) was destined to train (80% of the images) and test (20% of the images) the network, it offered a wide range of *P. oceanica* and environmental conditions, ensuring the robustness of the network training.

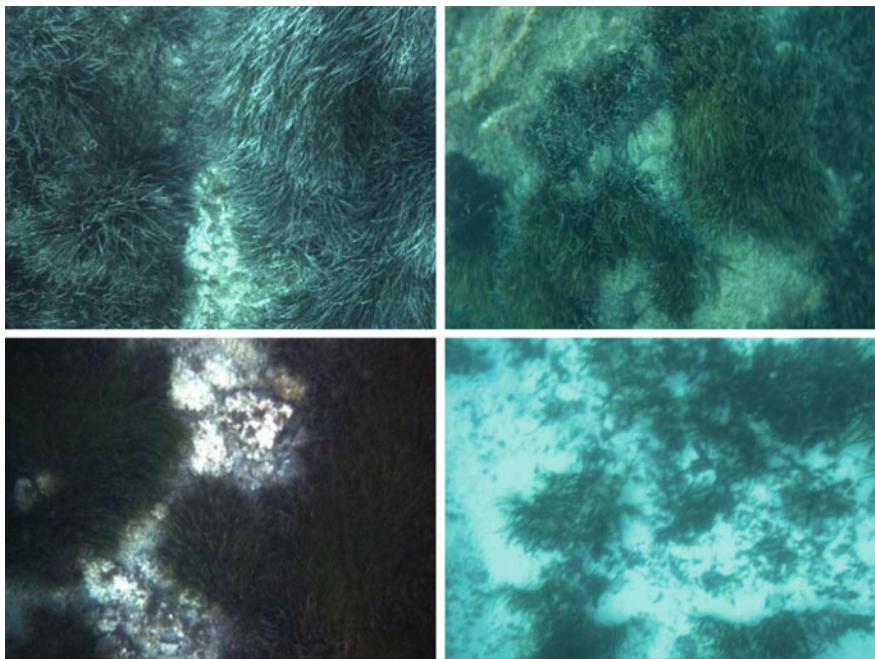
The *extra* dataset (23 images) was recorded using a different camera form the *mix* dataset. The *extra* dataset was used as an additional test. It helps to detect training overfitting, providing information on how well the network generalises its training on images containing distinct unseen conditions (camera used and *P. oceanica* or environmental conditions).



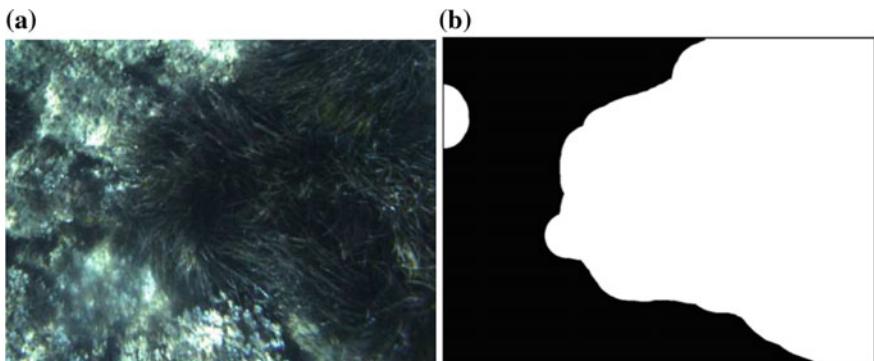
**Fig. 3** Map of the study area showing the island of Mallorca in the Western Mediterranean. Sampling points are indicated with arrows

**Table 1** Dataset arrangement

Mission	Location	Month	Camera	No. Im.	Set
1	Palma Bay	March	Manta G283	164	<i>mix</i>
2	Cala Blava	August	Manta G283	30	<i>mix</i>
3	Valldemossa	November	GoPro	157	<i>mix</i>
4	Valldemossa	October	Manta G283	68	<i>mix</i>
5	Valldemossa	September	Manta G283	41	<i>mix</i>
6	Valldemossa	June	BumbleBee2	23	<i>extra</i>



**Fig. 4** *Posidonia oceanica* images presenting a variety of *P. oceanica* and environmental conditions



**Fig. 5** **a** Original image. **b** Label map

**Table 2** Hyperparameter combination

Index	Data aug.	Learning rate	Iterations
1	0	1e-05	8
2			16k
3		5e-04	8k
4			16k
5	1	1e-05	8k
6			16k
7		5e-04	8k
8			16k

## Hyperparameter Combination

In order to find the hyperparameters that offer the best performance, the network was trained with the different values and combinations, shown in Table 2.

First, the network was trained with and without implementing data augmentation, this technique consists in applying contrast, brightness, color and morphological transformations to the training images in order to train over more diverse data, helping to reduce overfitting [48]. Secondly, two different learning rates were set, modifying the training step size when minimizing the loss [49]. Finally, two number of iterations were used, setting the times the network backpropagates and trains [49].

## Experiments

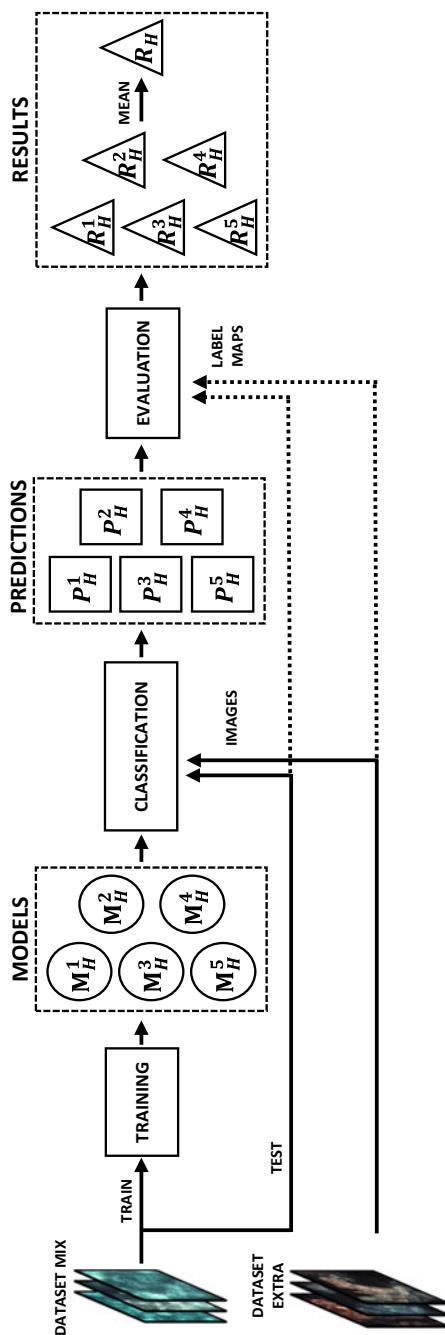
Following the methodology explained in Sect. 2, eight different experiments were conducted  $K = 1, 2, \dots, 8$ , each one assessing the performance of a hyperparameter combination, using its corresponding hyperparameters and applying a 5 k-fold cross-validation  $i = 1, 2, \dots, 5$ . On each cross validation, 4 subsets of the mix dataset (80% of the data) were used to train the network and the remaining one (20% of the data) was used to test it. Also, the entire extra dataset was used to test the network. This process is described in Fig. 6.

The evaluation process of each model starts by binarizing its probabilistic outputs, we decided to perform this binarization at nine equally distributed threshold values,  $j = 1, 2, \dots, 9$  (Fig. 7).

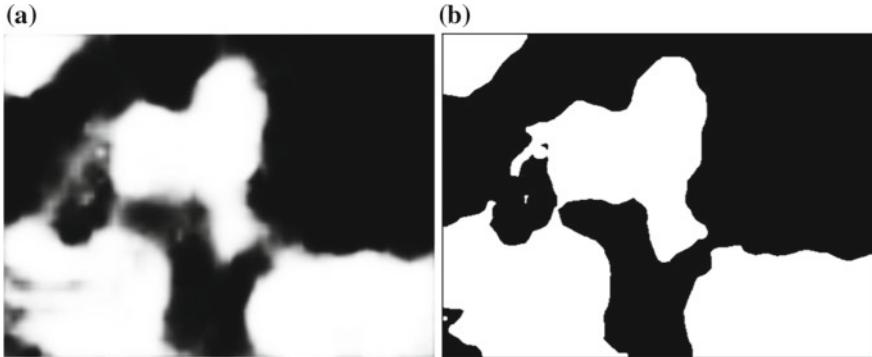
Then, we preformed a comparison between each binarized output and its corresponding label maps, acting as ground truth.

From this comparison, we generated confusion matrix, which indicates the number of *P. oceanica* pixels identified correctly (*True Positives*, TP) and wrongly (*False Positives*, FP), and also the number of background pixels identified correctly (*True Negatives*, TN) and wrongly (*False Negatives*, FN). From these values, the *accuracy*, *precision*, *recall* and *fall-out* of the model are computed.

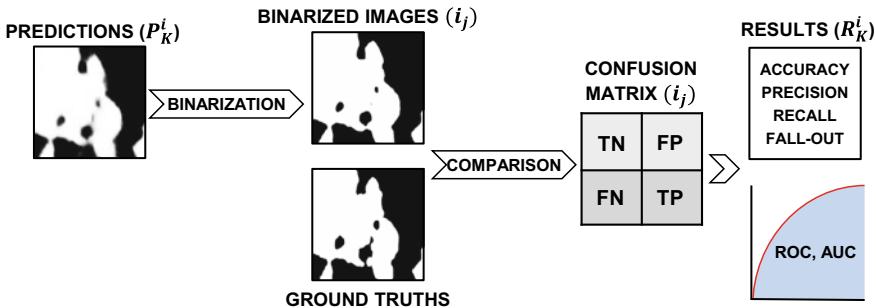
Finally, a *Receiver Operating Characteristic* (ROC) curve is generated [50], representing the *recall* against *fall-out* values of the classifier at various thresholds. The



**Fig. 6** Experiment “K” validation process. For each study cases, the network was trained five times using the k-fold cross validation method, generating five models. Each model was evaluated on the *mix* test data and *extra* whole dataset. The final experiment results were obtained as the mean performance of its five models



**Fig. 7** Probabilistic network output of an image (a) and one of its corresponding binarizations (b)



**Fig. 8** Evaluation process for the model “i” of experiment “K”. The network prediction is binarized at  $j = 1, 2, \dots, 9$  threshold values, generating a confusion matrix for each one. Finally, the evaluation metrics are calculated

analysis of the *Area Under the Curve* (AUC) of the ROC curve offers measure of the classifier performance.

Figure 8 represents the process followed to evaluate a model.

### 3.3 Classification Results

This section presents the obtained results for each experiment along with the hyper-parameter selection process.

In this section we use a three digit annotation to refer to each experiment, indicating its hyperparameters. The first digit implies if data augmentation was used (1) or not (0). The second one indicates if the used learning rate is  $1e-5$  (1) or  $5e-4$  (5). The last digit indicates if the number of iterations is 8000 (8) or 16,000 (16).

## Experiment Performance

### Mix Dataset Results

Figure 9 shows the results of evaluating the *mix* test set. In Fig. 9a, the ROC curve along its AUC value for each experiment is presented. While in the the *precision* and *accuracy* values at the optimal binarization threshold are represented in Fig. 9b in bar charts. The optimal binarization threshold is selected as the one presenting higher *trade-off* between *recall* and *fall-out*, calculated as:

$$\text{Trade-off} = \frac{\text{Recall} + (1 - \text{Fall-out})}{2} \quad (1)$$

The ROC curves for all experiments showed AUC values over 95%, reaching a maximum of 98.7% for the 1\_1\_16 experiment. According to the criteria established in [51] to determine how good a classifier is based on its AUC value, these values represent excellent classifiers.

*Precision* and *accuracy* values were greater than 90% for all the experiments. The maximum *Precision* achieved was 97.5%, for the experiment 1\_1\_8, while the lowest one was 92.2%. For the *accuracy*, the maximum achieved was 96.5%, for the experiment 1\_1\_16.

The comparison of the different experiments on a hyperparameter basis showed that:

- Experiments with lower learning rates presented better *precision*, *accuracy* and AUC values than experiments with higher rates.
- The effect of the number of iterations is almost negligible, being the metrics slightly better when trained over 16,000 iterations.
- The application of data augmentation had a similar slight effect than the number of iterations, presenting a small benefit when it was applied.

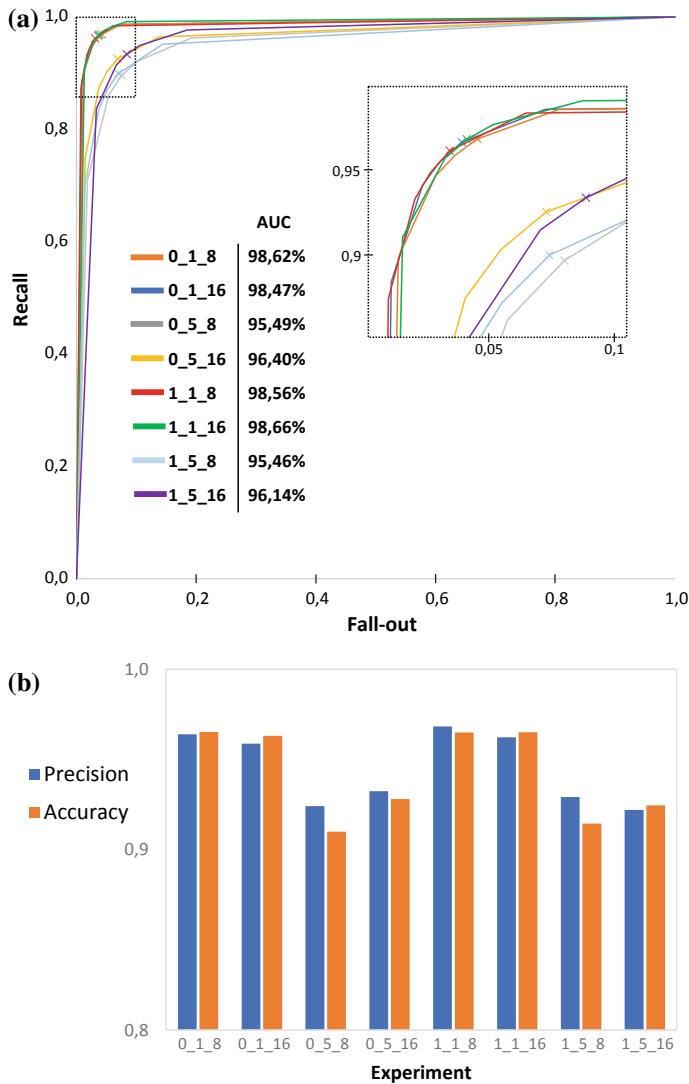
These almost negligible effects may be due specific conditions of our application, such as the network already being trained after the 8k iterations, and the train set already being diverse on its own, respectively.

Figure 10 shows qualitative results over images of the *mix* test set.

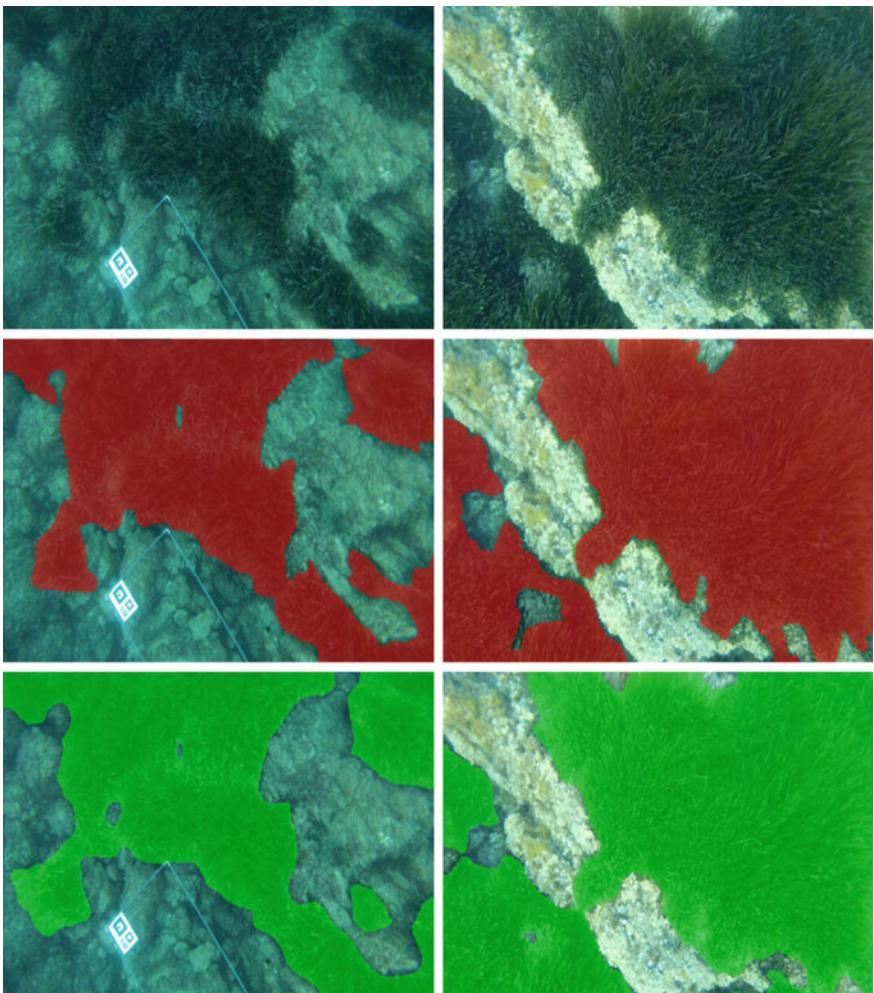
### Extra Dataset Results

The results obtained on the *mix* dataset were promising but, as mentioned in Sect. 3.2, the test images were extracted from the same immersions used to train the network, containing similar environmental conditions. To assess the performance of each model on unseen conditions, we evaluated them over the extra dataset, the results are presented in Fig. 11.

The AUC value of experiments that used a learning rate of 5e-04 were lower than the ones achieved in the *mix* test set evaluation results, reaching values around 92%. On the other hand, experiments that used a learning rate of 1e-05 were able to maintain the good results obtained on the *mix* dataset, achieving AUC values around 97.7% when the network was trained for 16,000 iterations and 97.0% when 8000.



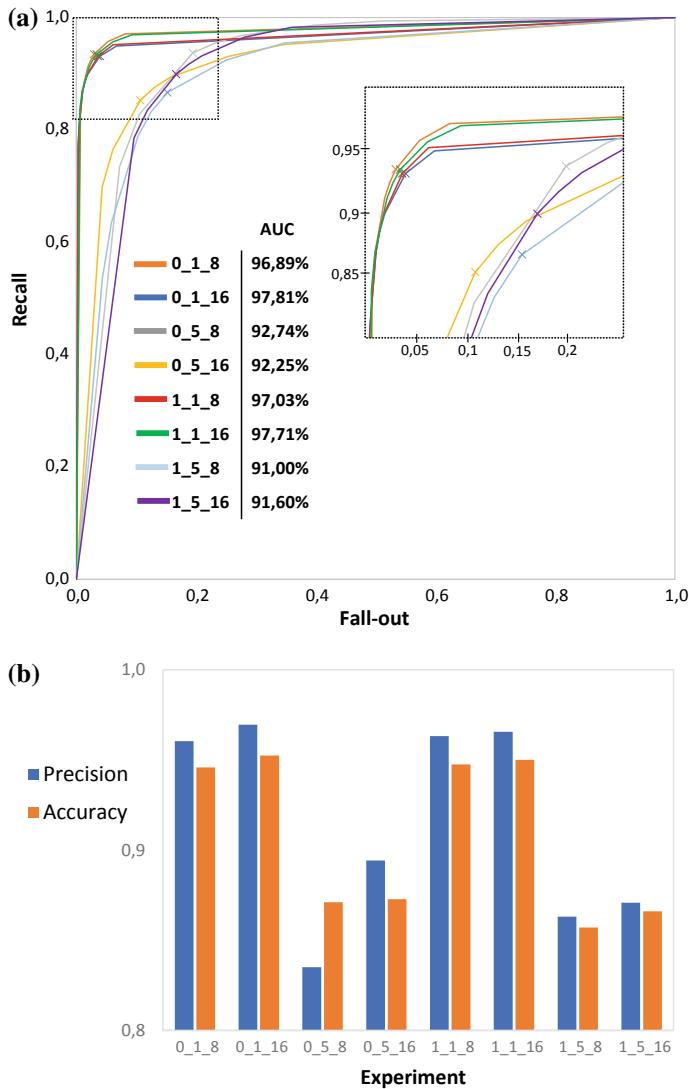
**Fig. 9** Mix test set results. **a** ROC curves and corresponding AUC. **b** Precision and accuracy metrics obtained at the optimal binarization threshold



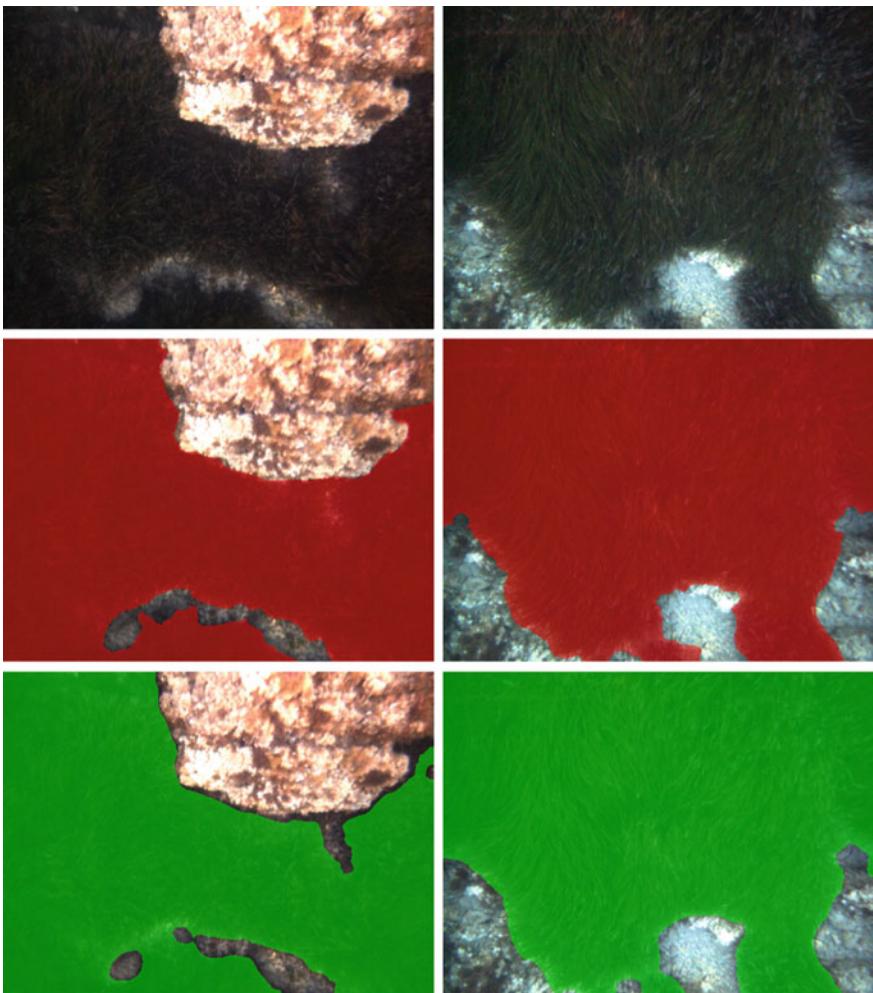
**Fig. 10** Qualitative results obtained for images from the *mix* test set. On the first row, two original images are shown. The second row of images illustrate the original images with their corresponding ground truth superimposed in red. Finally, the last set of images show the results of the segmentation superimposed in green to the original images

These results show that the models do not overfit the training images, being able to generalize its training to images taken with a different camera, containing different unseen environmental and *P. oceanica* conditions.

The same trend can be seen for the *precision* and *accuracy*, where experiments with higher learning rates achieved values for both metrics around 85%, while experiments that used lower learning rates only achieved values around 96% and 95%,



**Fig. 11** Extra test set results. **a** ROC curves and corresponding AUC. **b** Precision and accuracy metrics obtained at the optimal binarization threshold



**Fig. 12** Qualitative results obtained for images from the *extra* dataset. On the first row, two original images are shown. The second row of images illustrate the original images with their corresponding ground truth superimposed in red. Finally, the last set of images show the results of the segmentation superimposed in green to the original images

respectively. It also can be seen, experiments where the number of iterations was set to 16,000 presented slightly higher metrics.

Figure 12 shows qualitative results over test images of the *extra* dataset.

#### Hyperparameters Evaluation

We conducted an overall comparison on a hyperparameter basis from the evaluation results of all experiments, finding the hyperparameters which offer a better performance.

The results clearly indicate that experiments that used lower learning rates obtained better AUC, *precision* and *accuracy* results. The best learning rate was identified at 1e-05. Also, it can be seen that experiments conducted using a large number of steps tend to have a slightly better performance. The best number of iterations was identified at 16,000. Finally, we decided to apply data augmentation, helping to generalize the training to new unseen conditions for future immersions.

## 4 Jellyfish Detection and Identification

Over the past decades the social and scientific concern about increasing jellyfish numbers has risen. This can be noticed on the number of reports on jellyfish, over the past two decades the number of media and news reports have dramatically increased by over 500% [52], often with alarmist headlines [53].

Parallel to this, there is an ongoing scientific debate on whether jellyfish numbers are on the rise, on the one hand, some scientists argue that populations are increasing due to a range of natural and man-made causes [54, 55], while on the other, some scientists defend that jellyfish populations have remained constant over time [53]. The lack of base line data to endorse conclusions makes it difficult to support either argument.

Regardless, of the outcome of the debate, coastal populations are increasing, with 40% of the global population living within 100 km of the coast [56] and many more spending their holidays and free time in coastal areas. The increase in the use of the coast and its associated resources and benefits is leading to a higher rate of encounters between humans and jellyfish with all the associated socioeconomic consequences [57]. Among others, jellyfish aggregations are known to negatively affect coastal tourism with associated impacts on tourism revenues and the tourism industry [58]. Large aggregations of jellyfish can interfere with fishing operations by presenting a health hazard to fishermen when pulling the fishing gear on board, splitting the fishing nets due the weight of the jellyfish in the nets or ruining the catch [59]. In aquaculture, large aggregations of jellyfish have reportedly killed fish in pens [60, 61]. Water desalination and power plants have also suffered the consequences of the presence of high numbers of jellyfish, which can clog seawater intake screens causing power reductions and shutdowns [62, 63].

There is, therefore, a need to develop new technologies that enable the automatic detection of these organisms to facilitate the design of adaptive management strategies in order to mitigate jellyfish associated impacts. Furthermore, the development of such technology will greatly facilitate the collection of long term monitoring data in a cost-effective way.

So far, most studies aimed at monitoring and assessing the presence of jellyfish have relied on manual methods, such as visual countings from boats [64] or small aircrafts [65], or on a combination of video recording with subsequent human-based manual counting [13]. Manual methods, however, greatly limit the scope of the studies both from a spatial and a temporal perspective.

Automatic computer aided image classification represents a milestone for observational ecological studies [17], as it can deal with the aforementioned limitations. In Korea and Japan, where the presence of aggregations of big specimens of jellyfish often interfere with human uses of the coast, the first attempts based on the automatic detection of objects in images have been made to counteract the presence of these organisms. In Korea, an automatic jellyfish elimination system was created whereby an unmanned aerial vehicle equipped with cameras would identify jellyfish at the surface of the water and would eliminate them using a blade system [66]. In Japan, the first attempts to develop an automatic jellyfish detection system are underway, however so far the system has only been tested with artificially generated jellyfish images [67].

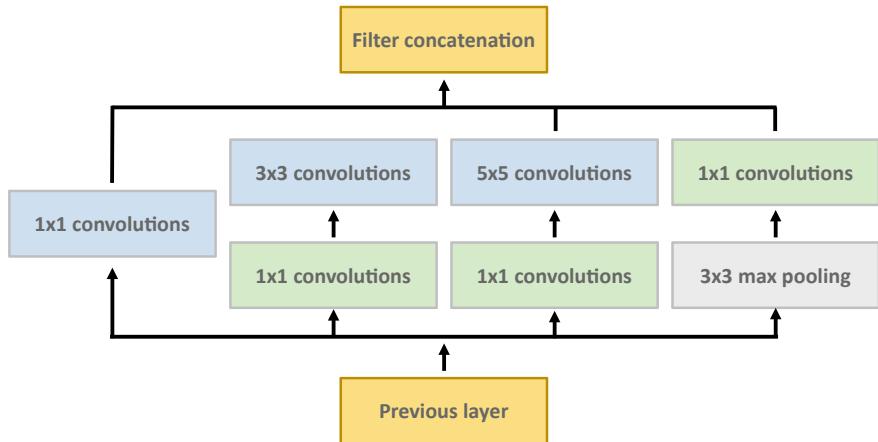
Here, we introduce an application of Deep Learning techniques for the detection of jellyfish. Deep Learning techniques have emerged as a promising methodology to enable the automatic detection and quantification of jellyfish, simultaneously allowing for the development of early warning systems for the presence of jellyfish. As an example, the analysis of video data through the identification and quantification of jellyfish can be used in coastal areas to detect jellyfish abundances and decide the optimal point for beach closures in order to avoid undesired socioeconomic effects. From a scientific perspective, the use of automatic detection techniques will permit the creation and maintenance of much needed long-term data series in a cost-effective way.

The case study is located on the island of Mallorca (Balearic Islands, Western Mediterranean basin) (Fig. 3). Mallorca is one of the major tourism destinations in Europe where the presence of jellyfish can sometimes cause undesired effects on tourism satisfaction. The example focuses on three of the most commonly encountered jellyfish species in Balearic waters, namely *Pelagia noctiluca*, *Cotylorhiza tuberculata* and *Rhizostoma pulmo* (Fig. 17). *P. noctiluca* can become very abundant during spring and summer months and has a fairly painful sting that can be very off-putting. *C. tuberculata* although abundant towards the end of the summer is inoffensive to humans. Finally, *R. pulmo* is mildly stinging, however due to its relatively big size (up to 40 cm in radius) and solid appearance, swimmers are generally able to spot it before getting stung.

The following sections explain the deep network architecture used and its characteristics, the different case studies, data processing, model tuning and validation process, and finally, the classification results.

## 4.1 Deep Learning Approach

In this application an object detection architecture is used for the detection and classification of the different jellyfish species. In the following section, the network architecture and the training details are presented.



**Fig. 13** Inception module, showing how the input is convoluted by three different kernel sizes:  $1 \times 1$ ,  $3 \times 3$ , and  $5 \times 5$ . To limit the number of input channels, an extra  $1 \times 1$  convolution is added before the  $3 \times 3$  and  $5 \times 5$  convolutions

## Network Architecture

The architecture used is the Inception-Resnet V2 [68], a very deep convolutional neural network with over 450 layers that it can efficiently learn to identify objects on images, outputting instance bounding boxes and classifying them into one of the specified classes with a confidence percentage.

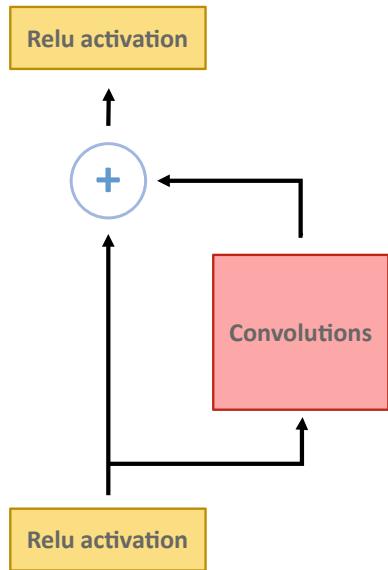
When detecting objects on an image, one of the main problems is to select the kernel sizes for the convolutional layers, as the same object may appear with huge size and shape variations from one instance to another. A larger kernel is preferred for bigger, more global instances, and a smaller kernel is preferred for smaller ones. To tackle this issue, the architecture performs multiple parallel convolutions using different kernel sizes, making the network “wider” rather than “deeper”. The blocks of layers containing these convolutions are called inception modules [69], represented in Fig. 13.

Another characteristic of the network, is the use of Residual Connections [70], used to add the output of the convolution operation of the inception module to the input. This introduces shortcuts in the model and it translates into a more optimal and accurate network. Figure 14 shows the structure of a Residual Connection.

This architecture combines the inception modules with Residual Connections, obtaining the so called Inception-ResNet modules. Figure 15 shows an example of these modules.

With these Inception-ResNet Modules, the main body of the architecture is built. Figure 16 shows a compressed view of the architecture.

**Fig. 14** Residual connection structure



## Training Details

The Inception-ResNet V2 architecture is trained by means of readjusting the values of the kernels in the convolutional layers, backpropagating the loss computed over the predictions obtained on the softmax layers.

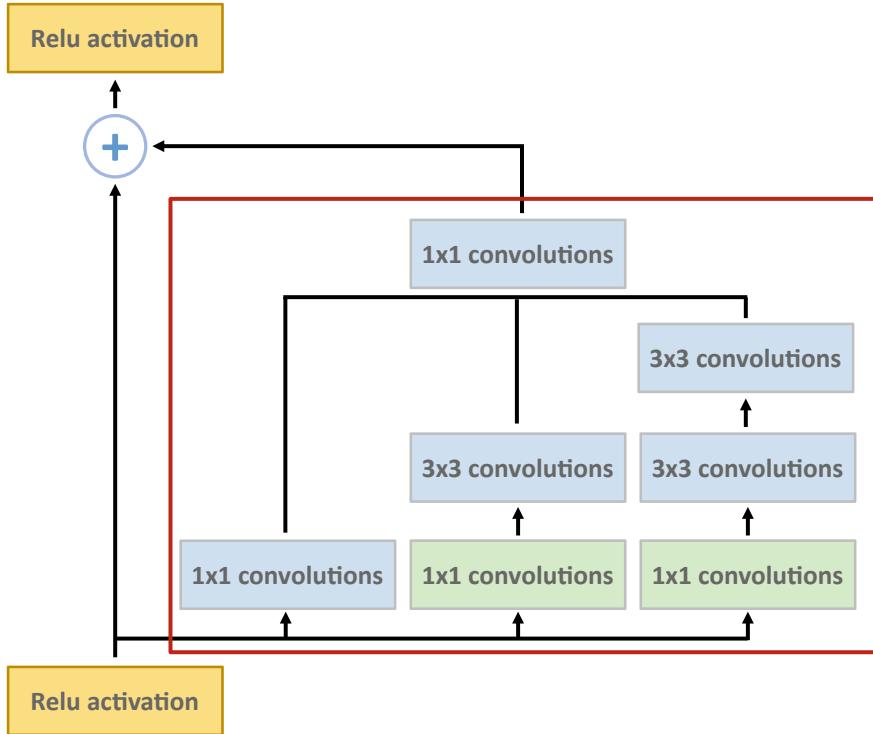
Due to the high number of layers, the loss becomes small and insufficient to update the kernel values properly. To prevent the middle part of the network from “dying out” during the backpropagation process, an auxiliary classifier is applied at the output of the second block of Inception-ResNet modules. In this way, an auxiliary loss is computed and added to the prior one as shown in Eq. 2.

$$\text{Total\_loss} = \text{main\_loss} + \text{aux\_loss} \times 0.3 \quad (2)$$

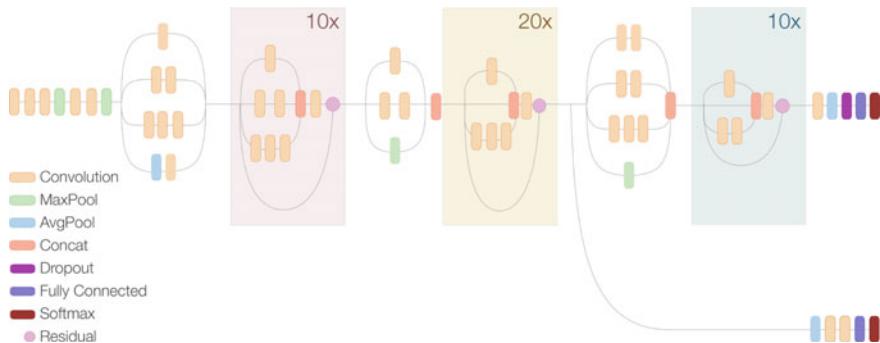
In order to train the network and adjust the kernel weights, a backpropagation function is needed. For this case, a smooth L1 location loss function is used, its loss increases as the predicted bounding box location diverges from the one specified on the ground truth. Also, the Momentum optimiser algorithm along with gradient clipping strategies [71] are implemented in order to help the training process reach the global minimum error.

The architecture used for this application, had already been trained over the COCO dataset [72]. To retrain the network with the desired classes, a set of images containing different jellyfish species and its corresponding ground truth are needed. The ground truth in this case is a text file for each image, where the bounding box and class for each jellyfish instance present in the image are indicated.

The trainings were performed on the same computer mentioned in Sect. 3.1.



**Fig. 15** Inception-ResNet-A module. The Max pooling branch from the Inception Module is substituted by the Residual Connection. The  $5 \times 5$  convolution is split into two equivalent  $3 \times 3$  convolutions, boosting computer and accuracy performance (neural networks perform better when convolutions do not alter the dimensions of the input drastically). Finally, for the residual sum to work, the input and output after convolution must have the same dimensions, hence, a  $1 \times 1$  convolution is applied after the original convolutions, to match the depth sizes



**Fig. 16** Neural network architecture, mainly composed by Inception-ResNet-A/B/C Modules, along with other complementary modules. More in depth information about this architecture can be found in [68]

## 4.2 Experimental Framework

This section describes the experimental framework followed. First, the image acquisition, organisation and labelling processes are described. Subsequently, the different case studies and hyperparameters used are presented. Finally, we describe the validation and evaluation details.

### Datasets

#### *Acquisition*

Training and testing images were extracted from underwater video sequences of the three species under consideration. The objective was to construct a dataset containing the three species under different conditions, such as water coloration, turbidity, illumination and different jellyfish positions and sizes, assuring robustness in the training process.

A dataset of 842 images was generated, 80% of the dataset was used to train the network (674 images), while the remaining 20% was used for testing purposes (168 images).

Figure 17 shows sample images from the dataset showcasing different conditions.

#### *Labelling*

For every image of the dataset, an annotation file was generated using the LabelImg tool [73], this generates an “.xml” file which contains the position and classification of each instance present in the image. Figure 18 shows an original image along with its ground truth “.xml” text file.

### Case Studies

Following the same procedure used in the previous application in Sect. 3.2, the network was trained using different sets of hyperparameters. The network was first trained with and without implementing data augmentation, secondly, two different learning rates were set, and finally, the network was trained using two values for the number of iterations. Results showing the different combinations of hyperparameters are shown in Table 3.

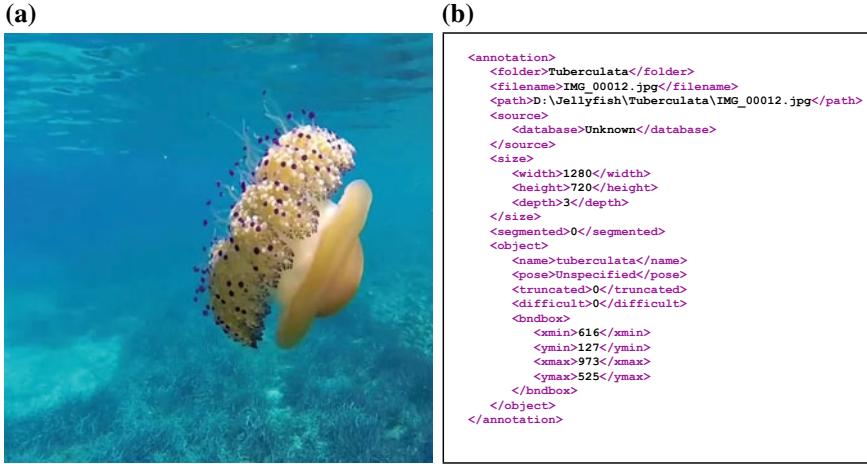
### Experiments

Following the methodology described in Sect. 2 and implemented in the previous application in Sect. 3.2, twelve different experiments were conducted  $K = 1, 2, \dots, 12$ , each one assessing the performance of a case study, using its corresponding hyperparameters and applying a 5 k-fold cross-validation  $i = 1, 2, \dots, 5$ , as shown in Fig. 19.

In order to evaluate the performance of each model, the Intersection over Union (IoU) method along with the average precision metric (AP) [74] were used, these are the most common evaluation methods for object detection, used in object detection competitions such as PASCAL VOC [75], ImageNet [76] or COCO [72].



**Fig. 17** Images from the dataset showing the three jellyfish species under different environmental conditions. Top: *P. noctiluca*, centre: *R. pulmo*, bottom: *C. tuberculata*



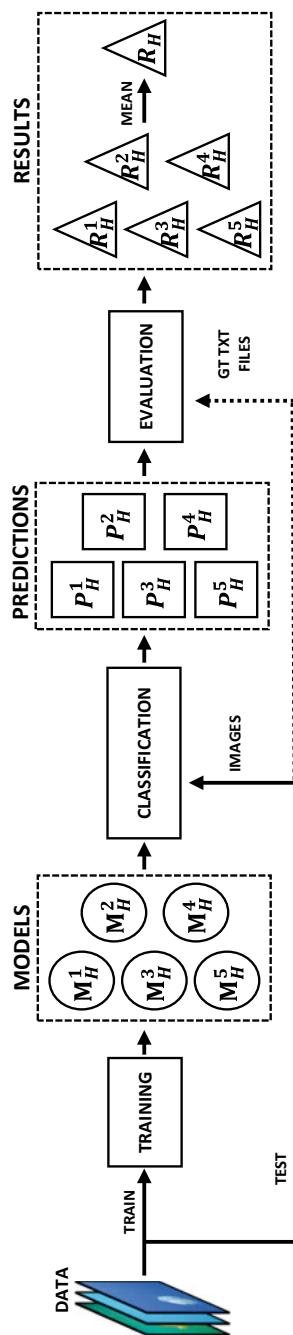
**Fig. 18** **a** Original image. **b** Corresponding ground truth “.xml” file, specifying the jellyfish location and class

**Table 3** Case studies. When applying data augmentation, random rotations and horizontal and vertical flips are applied. The decay learning rate consists in applying a learning rate of 5e-04 until 50% of the training and then dropping it to 5e-05

Case	Data aug.	Learning rate	Iterations (k)
1	0	5e-04	10
2			20
3			40
4		Decay	10
5			20
6			40
7	1	5e-04	10
8			20
9			40
10		Decay	10
11			20
12			40

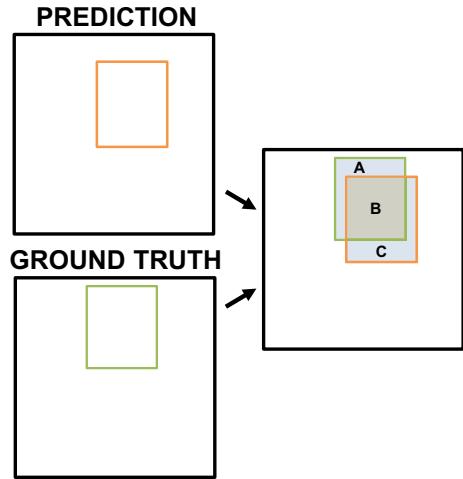
The IoU measure gives the similarity between the predicted and the ground-truth bounding-boxes areas, and is defined as the area of the intersection between bounding-boxes divided by the union of the bounding-boxes areas (see Eq. 3). Figure 20 illustrates how the IoU is calculated for a prediction.

$$IoU = \frac{A_{intersection}}{A_{union}} \quad (3)$$



**Fig. 19** Experiment “K” validation process. For each one of the twelve study cases, the network was trained five times using the k-fold cross-validation method. The output models were run and evaluated over the test data. The final experiment results were obtained as the mean performance of its five models

**Fig. 20** Representation of how the IoU is calculated between a prediction and a ground truth bounding-boxes. The IoU value would be calculated as:

$$IoU = \frac{B}{A + B + C}$$


Once the IoU is calculated for a prediction, in order to determine if that predictions is a TP or a FP, a threshold value over the IoU is established. Following the criteria applied in the PASCAL VOC challenge, this threshold is set at  $thr_{iou} = 0.5$ . A prediction is classified as TP if the IoU value with any ground truth bounding-box is greater than the  $thr_{iou}$  and the predicted class matches the corresponding one of the ground truth, otherwise, the detection is marked as a FP. The following equation represents this criteria:

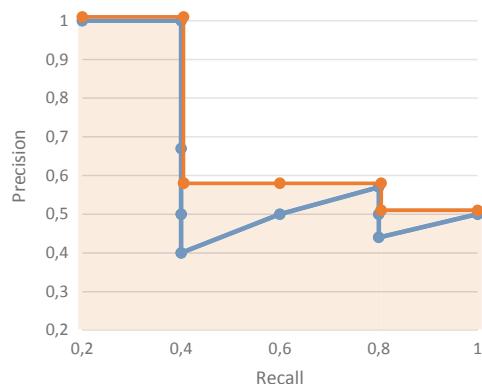
$$Detection = \begin{cases} TP, & \text{if } IoU \geq thr_{iou} \& C_{pred} = s_{gt} \\ FP, & \text{otherwise} \end{cases} \quad (4)$$

Also, ground truth instances which do not have a  $IoU > thr_{iou}$  with any prediction are marked as FN.

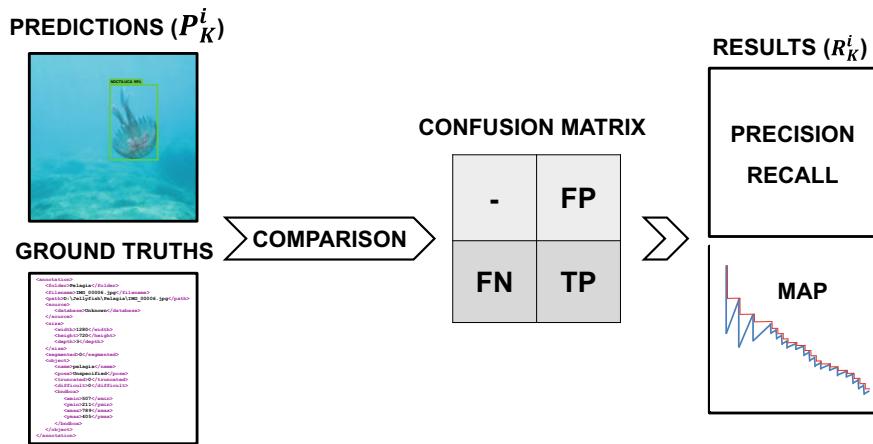
From the TP, FP and FN values, the *precision* and *recall* metrics are calculated for all classes. Finally, from these metrics, the AP of each class and mean AP (mAP) between classes are obtained. The AP can be understood as the average of the maximum precision at different recall values, or the area under *max(Precision)-Recall* curve. Figure 21 exemplifies calculus of the AP for a series of detections. More information about this evaluation metric can be found in [74].

The followed workflow to determine the detection performance of each model is showcased in Fig. 22.

Detection	TP/FP	Precision	Recall
1	TP	1	0,2
2	TP	1	0,4
3	FP	0,67	0,4
4	FP	0,5	0,4
5	FP	0,4	0,4
6	TP	0,5	0,6
7	TP	0,57	0,8
8	FP	0,5	0,8
9	FP	0,44	0,8
10	TP	0,5	1



**Fig. 21** Example calculus of the AP for a series of detections. The blue line represents the *precision-Recall* curve. The orange lie represents the *max(Precision)-Recall* curve. The AP value equals to the area under the *max(Precision)-Recall* curve (orange area)



**Fig. 22** Model "i" of experiment "K" evaluation process. The detection output is compared with its corresponding ground truth, obtaining the FP, FN and TP values. From these, the *Precision*, *Recall* and mAP values are calculated

### 4.3 Results

This section shows the results for each of the experiments and the assessment of the hyperparameter.

#### Experiment Performance

The mean results obtained when evaluating the five models of each experiment over their corresponding test set are shown in Table 4, showing the AP obtained for each class and the mAP value.

**Table 4** Results obtained for all experiments from evaluating the test set, showing the AP for each class and the mAP over all classes

Exp.	Data aug.	Learning rate	Iterations (k)	AP noctiluca (%)	AP pulmo (%)	AP tuberculata (%)	mAP (%)
1	0	0.0005	10	74.2	96.5	96.5	89.1
2			20	75.3	95.7	96.8	89.3
3			40	74.4	96.5	96.8	89.3
4		Decay	10	75.6	97.3	96.4	89.7
5			20	76.5	97.7	96.3	90.2
6			40	77.1	98.1	96.3	90.5
7	1	0.0005	10	72.9	98.1	96.6	89.2
8			20	76.4	97.7	96.7	90.3
9			40	<b>77.3</b>	98.0	96.6	<b>90.6</b>
10		Decay	10	73.7	98.5	96.9	89.7
11			20	74.8	<b>98.6</b>	<b>97.1</b>	90.1
12			40	76.2	98.5	96.8	90.5

All experiments show mAP values around 90%, reaching a maximum of 90.6% for experiment 9 and a minimum value of 89.1% for experiment 1. Looking at the AP values for the three species, it can be seen that both *R. pulmo* and *C. tuberculata* have much higher mAP values than *P. noctiluca*. This might be due to fact that *R. pulmo* and *C. tuberculata* are bigger specimens and the shape of their bodies remains relatively unchanged while swimming and therefore they might be easier to identify. On the contrary, in *P. noctiluca* the relative position of the tentacles in relation to the main body (umbrella) changes to a greater extent with the movement of the animal, adopting therefore a multitude of shapes and thus making it more difficult to identify.

Experiments where data augmentation is applied tend to have a slightly better performance. The same occurs with the number of iterations, experiments that are trained during 20 k or 40 k iterations show a small increase in performance. The application of the decay technique over the learning rate does not seem to have a significant impact over the performance. Qualitative results for the jellyfish detection process over the test dataset are shown in Fig. 23.



**Fig. 23** Visualization of the jellyfish detection obtained from images of the test set. The green bounding boxes represent *P. noctiluca* detections, the blue boxes correspond to *R. pulmo* and the orange ones to *C. tuberculata*

## 5 Conclusions

The two applications presented in this chapter clearly demonstrate the potential of deep learning in aiding the classification and processing of large image data sets collected in ecological studies. In the first study example, we showcase the application of a deep semantic segmentation neural network architecture to automatically detect the habitat forming seagrass species *P. oceanica*.

Diverse hyperparameter configurations were tested in order to find those that provided the best metrics. The evaluation results of the models showcased that the best metrics were achieved when data augmentation was applied and the network was trained for 16,000 iterations with a learning rate of 1e-05. A video presenting the network semantic segmentation can be seen at [77].

The results of this study are encouraging and show that deep learning techniques can be a useful tool for the automatic classification of underwater habitats. Future research should extend on this capability and build networks that can detect and classify multiple habitat types of coastal areas.

In the second example study, an object detection deep network has been used to automatically identify three commonly occurring species of jellyfish in the Mediterranean.

Once again, diverse hyperparameter configurations were tested in order to find those that provided the best metrics. In this case, the evaluation results of the models showcased that the best metrics were achieved when data augmentation was applied and the network was trained for 40,000 iterations with a decaying learning rate. A video presenting the network semantic segmentation can be seen at [78].

These results show the potential of object detection in the identification of marine species in image data, not only for jellyfish but for many other species that can be filmed in underwater environments. With respect to the detection of jellyfish, these results will be used to train the network to recognise more species. The demonstrated automatic detection methods will have direct applications for the monitoring of jellyfish in the proximity of beaches.

To conclude, deep learning techniques have a huge potential in supporting ecological studies. Once neural networks are functioning to a high precision in the detection of habitats or species, they can be applied to other datasets originating from other locations, thus providing help in image data processing for a much wider, potentially global, audience of scientists. Equally, with the help of experts in providing classified images, existing neural networks can be extended to include more habitats or species in the future. However, to further this development, information technologists and natural scientists alike need to more actively engage with each other fields and search for collaborations. Deep learning techniques have an essential part to play in moving ecological studies to a new level, providing more cost-effective data collection solutions at a time when large amounts of data are needed to detect and adapt to global environmental change.

**Acknowledgements** This work is partially supported by Ministry of Economy and Competitiveness (AEI, FEDER, UE), under contracts TIN2017-85572-P, DPI2017-86372-C3-1-R. H Hinz was

supported by the Ramón y Cajal Fellowship (grant by the Ministerio de Economía y Competitividad de España and the Conselleria d'Educació, Cultura i Universitats Comunidad Autónoma de las Islas Baleares). We would like to thank Charlotte Jennings for her help in identifying and labelling jellyfish in underwater images.

## References

1. Caughlan, L.: Cost considerations for long-term ecological monitoring. *Ecol. Indic.* **1**(2), 123–134 (2001)
2. Barrio Foján, C.R.S., Cooper, K.M., Bolam, S.G.: Towards an integrated approach to marine benthic monitoring. *Mar. Pollut. Bull.* **104**(1–2), 20–28 (2016)
3. Del Vecchio, S., Fantinato, E., Silan, G., Buffa, G.: Trade-offs between sampling effort and data quality in habitat monitoring. *Biodivers. Conserv.* **28**(1), 55–73 (2018)
4. Bennett, M., Acott, C., Richardson, K., Bowen, S., Smart, D., Smith, P., Sharp, F., Bryson, P., Goble, S.: Recreational technical diving part 1: an introduction to technical diving methods and activities. *J. S. Pac. Underw. Med. Soc. Eur. Underw. Baromedical Soc.* **43**(4), 86–93 (2013)
5. Hissmann, K., Schauer, J.: Manned submersible JAGO. *J. Large-Scale Res. Facil. JLSRF* **3**, A110 (2017)
6. Sagalevich, A.M.: 30 years experience of Mir submersibles for the ocean operations. *Deep Sea Res. Part II Top. Stud. Ocean.* **155**(2017), 83–95 (2017)
7. Rosenkranz, G.E., Byersdorfer, S.C.: Video scallop survey in the eastern Gulf of Alaska, USA. *Fish. Res.* **69**(1), 131–140 (2004)
8. Lambert, G.I., Jennings, S., Hinz, H., Murray, L.G., Parrott, L., Kaiser, M.J., Hiddink, J.G.: A comparison of two techniques for the rapid assessment of marine habitat complexity. *Methods Ecol. Evol.* **4**(3), 226–235 (2013)
9. Santana-Garcon, J., Newman, S.J., Harvey, E.S.: Development and validation of a mid-water baited stereo-video technique for investigating pelagic fish assemblages. *J. Exp. Mar. Biol. Ecol.* **452**, 82–90 (2014)
10. Gallo, N.D., Cameron, J., Hardy, K., Fryer, P., Bartlett Douglas, H., Levin Lisa, A.: Submersible- and lander-observed community patterns in the Mariana and New Britain trenches: influence of productivity and depth on epibenthic and scavenging communities. *Deep Sea Res. Part I Ocean. Res. Pap.* **99**, 119–133 (2015)
11. Sheehan, E., Vaz, S., Pettifer, E., Foster, N., Nancollas, S., Cousens, S., Holmes, L., Facq, J.-V., Germain, G., Attrill, M.: An experimental comparison of three towed underwater video systems using species metrics, benthic impact and performance. *Methods Ecol. Evol.* **7**(07) (2016)
12. Langlois, T.J., Harvey, E.S., Fitzpatrick, B., Meeuwig, J.J., Shedrawi, G., Watson, D.L.: Cost-efficient sampling of fish assemblages: comparison of baited video stations and diver video transects. *Aquat. Biol.* **9**(2), 155–168 (2010)
13. Holmes, T.H., Wilson, S.K., Travers, M.J., Langlois, T.J., Evans, R.D., Moore, G.I., Douglas, R.A., Shedrawi, G., Harvey, E.S., Hickey, K.: A comparison of visual- and stereo-video based fish community assessment methods in tropical and temperate marine waters of Western Australia. *Limnol. Ocean. Methods* **11**(7), 337–350 (2013)
14. Martin-Abadal, M., Guerrero-Font, E., Bonin-Font, F., Gonzalez-Cid, Y.: Deep semantic segmentation in an AUV for online posidonia oceanica meadows identification. *IEEE Access* **6**, 60956–60967 (2018)
15. Gray, P.C., Fleishman, A.B., Klein, D.J., McKown, M.W., Bézy, V.S., Lohmann, K.J., Johnston, D.W.: A convolutional neural network for detecting sea turtles in drone imagery. *Methods Ecol. Evol.*, 1–11 (2018). In Review
16. Kotta, J., Valdivia, N., Kutser, T., Toming, K., Rätsep, M., Orav-Kotta, H.: Predicting the cover and richness of intertidal macroalgae in remote areas: a case study in the Antarctic Peninsula. *Ecol. Evol.* **8**(17), 9086–9094 (2018)

17. Wäldchen, J., Mäder, P.: Machine learning for image based species identification. *Methods Ecol. Evol.* **9**(11), 2216–2225 (2018)
18. Weinstein, B.G.: Scene-specific convolutional neural networks for video-based biodiversity detection. *Methods Ecol. Evol.* **9**(6), 1435–1441 (2018)
19. Willi, M., Pitman, R.T., Cardoso, A.W., Locke, C., Swanson, A., Boyer, A., Veldthuis, M., Fortson, L.: Identifying animal species in camera trap images using deep learning and citizen science. *Methods Ecol. Evol.*, 1–12 (2018)
20. Mieszkowska, N., Sugden, H., Firth, L.B., Hawkins, S.J.: The role of sustained observations in tracking impacts of environmental change on marine biodiversity and ecosystems. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **372**(2025) (2014)
21. Gouraguine, A., Moranta, J., Ruiz-Frau, A., Hinz, H., Reñones, O., Ferse, S.C.A., Jompa, J., Smith, D.J.: Citizen science in data and resource-limited areas: a tool to detect long-term ecosystem changes. *Plos One* **14**(1), e0210007 (2019)
22. Cun, L., Cun, L., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. *Adv. Neural Inf. Process. Syst.* **2**, 396–404 (1990)
23. Russakovsky, O., Deng, J., Hao, S., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis.* **115**(3), 211–252 (2015)
24. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. [ArXiv:1409.1556](https://arxiv.org/abs/1409.1556), Sept 2014
25. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. [ArXiv:1512.03385](https://arxiv.org/abs/1512.03385), Dec 2015
26. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely Connected Convolutional Networks, Aug 2016
27. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. [ArXiv:1409.4842](https://arxiv.org/abs/1409.4842), Sept 2014
28. Chollet, F.: Xception: deep learning with depthwise separable convolutions. [ArXiv:1610.02357](https://arxiv.org/abs/1610.02357), Oct 2016
29. Geisser, S.: The predictive sample reuse method with applications. *J. Am. Stat. Assoc.* **70**(350), 320–328 (1975)
30. Diaz-Almela, E., Duarte, C.: Management of Natura 2000 Habitats 1120, (*Posidonia oceanicae*). Technical report, European Commission (2008)
31. Ruiz-Frau, A., Gelcich, S., Hendriks, I.E., Duarte, C.M., Marbà, N.: Current state of seagrass ecosystem services: research and policy integration. *Ocean. Coast. Manag.* **149**, 107–115 (2017)
32. Marbà, N., Duarte, C.: Mediterranean warming triggers seagrass (*posidonia oceanica*) shoot mortality. *Glob. Chang. Biol.* **16**(8), 2366–2375 (2010)
33. Telesca, L., Belluscio, A., Criscoli, A., Ardizzone, G., Apostolaki, E.T., Fraschetti, S., Gristina, M., Knittweis, L., Martin, C.S., Pergent, G., Alagna, A., Badalamenti, F., Garofalo, G., Gerakaris, V., Pace, M.L., Pergent-Martini, C., Salomidi, M.: Seagrass meadows (*posidonia oceanica*) distribution and trajectories of change. *Scientific reports* (2015)
34. y Royo, C.L., Pergent, G., Pergent-Martini, C., Casazza, G.: Seagrass (*posidonia oceanica*) monitoring in western mediterranean: implications for management and conservation. *Environ. Monit. Assess.* **171**, 365–380 (2010)
35. Sagawa, T., Komatsu, T.: Simulation of seagrass bed mapping by satellite images based on the radiative transfer model. *Ocean. Sci. J.* **50**(2), 335–342 (2015)
36. Montefalcone, M., Rovere, A., Parravicini, V., Albertelli, G., Morri, C., Bianchi, C.N.: Evaluating change in seagrass meadows: a time-framed comparison of side scan sonar maps. *Aquat. Bot.* **104**, 204–212 (2013)
37. Vasiljevic, A., Miskovic, N., Vukic, Z., Mandic, F.: Monitoring of seagrass by lightweight AUV: a *posidonia oceanica* case study surrounding Murter island of Croatia. In: Mediterranean Conference on Control and Automation, pp. 758–763, June 2014
38. Rende, F.S., Irving, A.D., Lagudi, A., Bruno, F., Scalise, S., Cappa, P., Montefalcone, M., Bacci, T., Penna, M., Trabucco, B., Di Mento, R., Cicero, A.M.: Pilot application of 3D underwater

- imaging techniques for mapping *Posidonia oceanica* (L.) delile meadows. *ISPRS - Int. Arch. Photogramm. Remote. Sens. Spat. Inf. Sci.*, 177–181 (2015)
- 39. Bonin-Font, F., Burguera, A., Lisani, J.-L.: Visual discrimination and large area mapping of *posidonia oceanica* using a lightweight AUV. *IEEE Access* **5**, 24479–24494 (2017)
  - 40. Gonzalez-Cid, Y., Burguera, A., Bonin-Font, F., Matamoros, A.: Machine learning and deep learning strategies to identify *posidonia* meadows in underwater images. *IEEE Oceans*, 1–5 (2017)
  - 41. Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. [arXiv:1603.07285](https://arxiv.org/abs/1603.07285), Mar 2016
  - 42. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, June 2015
  - 43. Teichmann, M., Weber, M., Zoellner, M., Cipolla, R., Urtasun, R.: MultiNet: real-time joint semantic reasoning for autonomous driving. [arXiv:1612.07695](https://arxiv.org/abs/1612.07695), Dec 2016
  - 44. Buja, A., Stuetzle, W., Shen, Y.: Loss functions for binary class probability estimation and classification: structure and applications. Technical report, Department of Statistics of University of Pennsylvania, Jan 2005
  - 45. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. [arXiv:1412.6980](https://arxiv.org/abs/1412.6980), Dec 2014
  - 46. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014)
  - 47. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Li, F.-F.: Imagenet: a large-scale hierarchical image database. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009
  - 48. Taylor, L., Nitschke, G.: Improving deep learning using generic data augmentation. [arXiv:1708.06020](https://arxiv.org/abs/1708.06020), Aug 2017
  - 49. Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. In: *Neural Networks: Tricks of the Trade* (2012)
  - 50. Provost, F., Fawcett, T., Kohavi, R.: The case against accuracy estimation for comparing induction algorithms. In: *Proceedings of the Fifteenth International Conference on Machine Learning*, Apr 2001
  - 51. Powers, D.M.W.: Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Int. J. Mach. Learn. Technol.* **2**, 37–63 (2011)
  - 52. European Commission: Assessment of jellyfish socioeconomic impacts in the mediterranean: implications for management, horizon 2020, May 2017
  - 53. Condon, R.H., Graham, W.M., Duarte, C.M., Pitt, K.A., Lucas, C.H., Haddock, S.H.D., Sutherland, K.R., Robinson, K.L., Dawson, M.N., Decker, M.B., Mills, C.E., Purcell, J.E., Malej, A., Mianzan, H., Uye, S.-I., Gelich, S., Madin, L.P.: Questioning the rise of gelatinous zooplankton in the world's oceans. *Bioscience* **62**(2), 160–169 (2012)
  - 54. Richardson, A.J., Bakun, A., Hays, G.C., Gibbons, M.J.: The jellyfish joyride: causes, consequences and management responses to a more gelatinous future. *Trends Ecol. Evol.* **24**(6), 312–322 (2009)
  - 55. Condon, R.H., Duarte, C.M., Pitt, K.A., Robinson, K.L., Lucas, C.H., Sutherland, K.R., Mianzan, H.W., Bogeberg, M., Purcell, J.E., Decker, M.B., Uye, S.-I., Madin, L.P., Brodeur, R.D., Haddock, S.H.D., Malej, A., Parry, G.D., Eriksen, E., Quinones, J., Acha, M., Harvey, M., Arthur, J.M., Graham, W.M.: Recurrent jellyfish blooms are a consequence of global oscillations. *Proc. Natl. Acad. Sci. USA* **110**(3), 1000–1005 (2013)
  - 56. United Nations: Factsheet: people and oceans, pp. 1–2. <https://www.un.org/sustainabledevelopment/wp-content/uploads/2017/05/Ocean-fact-sheet-package.pdf> (2017)
  - 57. Macrokanis, C., Hall, N., Mein, J.: Irukandji syndrome in Northern Western Australia: an emerging health problem. *Med. J. Aust.* **181**, 699–702 (2004)
  - 58. Fenner, P.J., Lippmann, J., Gershwin, L.A.: Fatal and nonfatal severe jellyfish stings in Thai waters. *J. Travel. Med.* **17**(2), 133–138 (2010)
  - 59. Purcell, J.E., Ichi Uye, S.-I., Tseng Lo, W.-T.: Anthropogenic causes of jellyfish blooms and their direct consequences for humans: a review. *Mar. Ecol. Prog. Ser.* **350**, 153–174 (2007)

60. Purcell, J.E., Baxter, E.J., Fuentes, V.L.: Jellyfish as products and problems of aquaculture. In: Advances in Aquaculture Hatchery Technology, pp. 404–430. Elsevier (2013)
61. Merceron, M., Le Fevre-Lehoerff, G., Bizouarn, Y., Kempf, M.: Fish and jellyfish in Brittany (France). *Equinoxe* **56**, 6–8 (1995)
62. Lee, J.H., Choi, H.W., Chae, J., Kim, D.S., Lee, S.B.: Performance analysis of intake screens in power plants on mass impingement of marine organisms. *Ocean. Polar Res.* **28**, 385–393 (2006)
63. Matsumura, K., Kamiya, K., Yamashita, K., Hayashi, F., Watanabe, I., Murao, Y., Miyasaka, H., Kamimura, N., Nogami, M.: Genetic polymorphism of the adult medusae invading an electric power station and wild polyps of *Aurelia aurita* in Wakasa Bay, Japan. *J. Mar. Biol. Assoc. UK* **85**(3), 563–568 (2005)
64. Ferraris, M., Berline, L., Lombard, F., Guidi, L., Elineau, A., Mendoza-Vera, J.M., Lilley, M.K.S., Taillandier, V., Gorsky, G.: Distribution of *Pelagia noctiluca* (Cnidaria, Scyphozoa) in the Ligurian Sea (NW Mediterranean Sea). *J. Plankton Res.* **34**(10), 874–885 (2012)
65. Barrado, C., Fuentes, J.A., Salamí, E., Royo, P., Olariaga, A.D., López, J., Fuentes, V.L., Gili, J.M., Pastor, E.: Jellyfish monitoring on coastlines using remote piloted aircraft. In: IOP Conference Series: Earth and Environmental Science, vol. 17, p. 12195 (2014)
66. Kim, Donghoon, Shin, J.U., Kim, H., Kim, H., Lee, D., Lee, S.M., Myung, H.: Development and experimental testing of an autonomous jellyfish detection and removal robot system. *Int. J. Control Autom. Syst.* **14**(1), 312–322 (2016)
67. Matsuura, F., Fujisawa, N., Ishikawa: Detection and removal of jellyfish using underwater image analysis. *J. Vis.* **10**(3), 259–260 (2007)
68. Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI Conference on Artificial Intelligence, Feb 2016
69. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9, June 2015
70. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
71. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on International Conference on Machine Learning, vol. 28, pp. III–1310–III–1318 (2013)
72. Lin, T.-Y., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Lawrence Zitnick, C.: Microsoft coco: common objects in context. In: ECCV (2014)
73. Tzutalin, D.: LabelImg. <https://github.com/tzutalin/labelImg> (2018)
74. Zhu, M.: Recall, precision and average precision. Technical report, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Canada, Sept 2004
75. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL visual object classes (VOC) challenge. *Int. J. Comput. Vis.* **88**(2), 303–338 (2010)
76. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: CVPR09 (2009)
77. Martin-Abadal, M.: Video: online posidonia oceanica segmentation. <http://srv.uib.es/pos-identification/>, May 2018
78. Martin-Abadal, M.: Video: jellyfish object detection. <http://srv.uib.es/jellyfish-object-detection/>, Dec 2018

# Deep Learning Case Study on Imbalanced Training Data for Automatic Bird Identification



Juha Niemi and Juha T. Tanttu

**Abstract** Collisions between birds and wind turbines can be significant problem in wind farms. Practical deterrent methods are required to prevent these collisions. However, it is improbable that a single deterrent method would work for all bird species in a given area. An automatic bird identification system is needed in order to develop bird species level deterrent methods. This system is the first and necessary part of the entirety that is eventually able to, monitor bird movements, identify bird species, and launch deterrent measures. The system consists of a radar system for detection of the birds, a digital single-lens reflex camera with telephoto lens for capturing images, a motorized video head for steering the camera, and convolutional neural networks trained on the images with a deep learning algorithm for image classification. We utilized imbalanced data because the distribution of the captured images is naturally imbalanced. We applied distribution of the training data set to estimate the actual distribution of the bird species in the test area. Species identification is based on the image classifier that is a hybrid of hierarchical and cascade models. The main idea is to train classifiers on bird species groups, in which the species resembles more each other than any other species outside the group in terms of morphology (coloration and shape). The results of this study show that the developed image classifier model has sufficient performance to identify bird species in a test area. The proposed system produced very good results, when the hybrid hierarchical model was applied to the imbalanced data sets.

**Keywords** Machine learning · Deep learning · Convolutional neural networks · Classification · Data augmentation · Intelligent surveillance systems

---

J. Niemi (✉) · J. T. Tanttu

Faculty of Information Technology and Communication Sciences, Tampere University Pori Unit,  
Pohjoisranta 11 A, 28101 Pori, Finland

e-mail: [juha.k.niemi@tuni.fi](mailto:juha.k.niemi@tuni.fi)

J. T. Tanttu

e-mail: [juha.tanttu@tuni.fi](mailto:juha.tanttu@tuni.fi)

## 1 Introduction

Demand for automatic bird identification systems for wind farms has increased recently. This kind of system is especially required for offshore wind farms. The objective of this application is twofold: it has first to detect two key bird species, which are particularly required for monitoring in the environmental license, and secondly to classify maximum number of other bird species while the first part still stands. The two key species are the white-tailed eagle (*Haliaeetus albicilla*) and the lesser black-backed gull (*Larus fuscatus fuscatus*). An automatic identification system is in development that consists of a separate commercial radar system to detect the birds, a digital single-lens reflex camera with telephoto lens for capturing images, a motorized video head for steering the camera, and a convolutional neural network (CNN) trained on the images with a deep learning algorithm for image classification. The conventional approach to this image classification problem is to presume that equally distributed data are fed into the classifier. However, this is a real-world application, in which it is difficult and time-consuming to collect large number of images for each class. Due to the nature of this application, it is conceivable that imbalanced data are utilized because the distribution of the captured images is naturally imbalanced, i.e., there are common and scarce bird species in the test area. It is also possible to include scarcer classes into the classification process with this approach. Researchers have proposed a class-imbalance aware loss function for the problem of class imbalance. This loss function adds an extra class-imbalance aware regularization term to the normal softmax loss [1]. However, we have applied the distribution of the training data set to estimate the actual distribution of the bird species in the test area. Training data set and test data set both share this distribution. Species identification is based on the image classifier that is a hybrid of hierarchical and cascaded models. The main idea is to train classifiers on bird species groups, in which the species resembles more each other than any other species outside the group in terms of morphology. The first classifier is hierarchical determining the group of the test image and the subsequent classifiers within the groups are in cascades. We have also applied our data augmentation method, which rotate and convert the images in accordance with the desired color temperatures. The hybrid hierarchical and cascade model is compared to two single classifiers. One of the classifiers is trained on balanced data set and the other is trained on imbalanced data set without grouping.

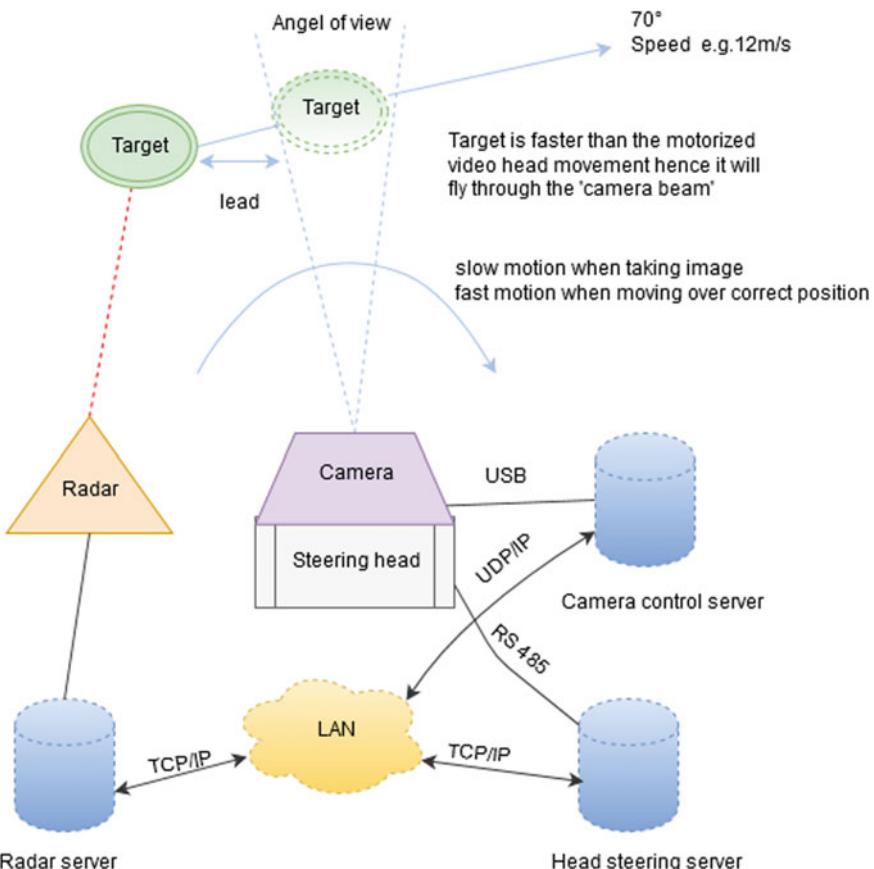
CNN has been successfully applied to image classification problems [2]. The number of training examples in image classification is typically large. This may cause problems when dealing with real-world applications, as collection of large number of images is not always possible. As a result, some data augmentation is usually needed [3, 4]. Cascade CNN has been successfully applied to face detection and road-sign classification system [5, 6].

The remainder of this chapter is organized as follows. In Sect. 2 we present the system and its components for collecting images automatically. In Sect. 3 related work is discussed. We describe our data, its grouping, and data augmentation algorithm in Sect. 4 Classification algorithms, applied CNN models, and feature extraction

are described in Sect. 5 Results for hybrid of hierarchical and cascade CNN model, trained on imbalanced data set and compared to conventional CNN model, are presented in Sect. 6. We then offer conclusions in Sect. 7.

## 2 The System

The proposed system consists of several hardware as well as software modules. See Fig. 1 for an illustration. At first, there is the radar system, which is connected to a local area network (LAN), and thus it is able to communicate with the servers, in which the various programs are running.



**Fig. 1** The hardware of the system and the principle of catching flying bird into the frame area of the camera

We use a radar system supplied by Robin Radar Systems B.V. because they provide an avian radar system that is able to detect birds. They also have algorithms for tracking a detected object over time (between the blips). The model we use is the ROBIN 3D FLEX v1.6.3 and it is actually a combination of two radars and a software package for implementation of various algorithms such as the tracker algorithms [7]. The role of the radar is to detect flying birds and pass the WGS84 coordinates of the target bird to the video head control software. The system includes the PT-1020 Medium Duty motorized video head of the 2B Security Systems. The video head is operated by Pelco-D control protocol [8], and the control software for it is developed by us. The System uses Canon EOS 7D II camera with 20.2-megapixel sensor and the Canon EF 500/f4 IS lens. Correct focusing of the images relies on the autofocus system of the lens and the camera. Automatic exposure is also applied. The camera is controlled by the application programmable interface (API) of the camera manufacturer, and the software for controlling the camera has been developed by us. In addition, the radar system provides parameters, which can be applied to increase the performance of classifiers. These parameters are the distance in 3D of a target (m), velocity of a target (m/s), and trajectory of a target (WGS84 coordinates). For the details of the system hardware, see [9].

### 3 Related Work

Researchers proposed a multi-sensor data fusion approach via acoustics, infrared camera, and marine radar for avian monitoring. The objective is to preserve the population of birds and bats especially those listed in endangered list, by observing their activity and behavior over the migration period. Species-level identification was not aimed mainly. They address to this problem by a fuzzy Bayesian based multi-sensory data fusion approach to provide the activity information regarding the targets in avian (birds and bats) monitoring [10].

Researchers have implemented machine learning (ML) algorithms on radar data for bird species classification. They used data collected from two locations in Portugal with two marine radar antennas (volume search radar, VSR and high sensitivity reception, HSR). The performance of six widely used ML algorithms: random forests (RF), support vector machine (SVM), artificial neural networks, linear discriminant analysis, quadratic discriminant analysis, and decision trees (DT), was tested. They found that all algorithms performed well (area under the receiver operating characteristic and accuracy  $>0.80$ ,  $P < 0.001$ ) when discriminating birds from non-biological targets such as vehicles, rain or wind turbines, but the algorithms showed greater variance in their performance when they classified different bird functional groups or bird species (e.g. herons vs. gulls). In this study, only RF was able to hold an accuracy  $>0.80$  for all classification tasks, although SVM and DT also performed well. All algorithms correctly classified 86% and 66% (VSR and HSR, respectively) of the target points, and only 2% and 4% of these points were misclassified by all algorithms. The results suggest that ML algorithms are suitable for classifying radar

targets as birds, and thereby separating them from other non-biological targets. The ability of these algorithms for correct identification between bird species functional groups was much weaker [11].

Time-lapse photography is a method in which the frame rate of taking a sequence of images is higher than the frame rate used to view the sequence. Time-lapse images can make subtle time-related processes distinct, and the process that is analyzed, can be too fast or too slow to the human eye. Time-lapse images have been used to detect birds around a wind farm. An Image-based detection using cameras have been applied to build a bird monitoring system. This system utilized an open-access time-lapse image data set that is collected around the wind farm. The system applied algorithms: AdaBoost, Haar-like, histogram of oriented gradients (HOG), and CNN. AdaBoost is a two-class classifier, which is based feature selection and weighted majority voting. A strong classifier is made as a weighted sum of many weak classifiers, and the resulting classifier is shallow but robust [12]. Haar-like is an image feature that utilizes contrasts in images. It extracts the light and the shade of objects by using black-and-white patterns [13]. HOG is a feature used for grasping the approximated shape of objects. At first, it computes the spatial gradient of the image and makes a histogram of the quantized direction of the gradient in each local region, called a cell in the image. Subsequently, it concatenates the histograms of the cells in the neighboring groups of the cells (the blocks) and normalizes them by dividing by their Euclidean norms in each block [14]. The best method for detection was Haar-like, and the best method for classification was CNN. The system was tested on only two bird functional groups, hawks and crows, and it achieved only moderate performance [15].

## 4 Data

Input data of this application consist of digital images. All images for training the CNN have been taken manually at the test location in various weather conditions. The location is the same where the camera will be installed for taking images automatically. The collected image set was divided into two data sets: an original data set for training classifiers, and a test set for measuring generalization of the classifiers, and thus the classifiers will not see these test images during training. Both data sets are divided into 14 classes. It became clear during image collection that there would be low number of images of the rarest bird species, resulting in classes with very low number of data examples. Therefore, in order to be able to classify the rarest bird species, all the collected images are included with an acceptance that the resulting data set will be imbalanced. The distribution of the number of images for each class is used as an estimate for the actual distribution of bird species in the test area. This is justified by the fact that images are collected in all four seasons and in all hours during day light. The estimate is not necessarily reliable in terms of bird species census, because only the species that usually fly at approximately same height with the wind turbines are taken into account, but it is sufficient in the context of this application.

The total number of images of the original data set is 24631, and number of images of the test data set is 439. The test data set was created by randomly choosing images from all classes. The number of images in the test data set follows the distribution of the original data set, thus reflecting the actual distribution in the test site. Class labels and number of images of the original data set for each class are presented in Table 1. In this Table, three classes are not defined in species level: LNSP, SWSP, and CATE. The first two cases are because there is no need to distinguish between loon species or swan species any further in this context, regardless the fact that two common, and two rare species of loons occur in the test area, and analogously there occur two common and one rare species of swans. The same applies to the third case too, the common/arctic tern. In addition, it is generally very difficult to tell the difference between these two tern species [16], and thus the number of required data examples (images) might be too large, considering the time needed to collect them. The number of images for each class in the test data set are also given in Table 1.

No preprocessing, other than cropping, is applied to the images before feeding them into the classifiers. The cropping is based on a segmentation, and it is motivated by being able to dispose the most of the pixels representing only sky. The resolution of the camera sensor measured by the total number of pixels and the focal length of the lens are important qualities because of the long range, of which images are to be taken. The effective number of pixels (ENP) is defined by the number of pixels representing a bird. The remaining number of pixels are considered noise, thus ENP has a significant effect on the performance of image classification model as birds will be very small (they consist of only a small number of pixels) in the images.

**Table 1** The original data set divided into 14 classes

# Images	# Test images	Class name (Eng.)	Class name (Lat.)	Class label
396	7	Loon Species	<i>Gavia</i> sp	LNSP
260	5	Swan Species	<i>Cygnus</i> sp	SWSP
5612	100	Great Cormorant	<i>Phalacrocorax carbo</i>	GRCO
979	17	Common Eider	<i>Somateria mollissima</i>	COEI
1164	21	Common Goldeneye	<i>Bucephala clangula</i>	COGO
236	4	Velvet Scoter	<i>Melanitta fusca</i>	VESC
263	5	Red-breasted Merganser	<i>Mergus serrator</i>	RBME
2450	44	White-tailed Eagle	<i>Haliaeetus albicilla</i>	WTEA
512	9	Great Black-backed Gull	<i>Larus marinus</i>	GBBG
4053	72	Herring Gull	<i>Larus argentatus</i>	HEGU
1481	26	Lesser Black-backed Gull	<i>Larus fuscus fuscus</i>	LBBG
3648	65	Common Gull	<i>Larus canus</i>	COGU
1803	32	Black-headed Gull	<i>Larus ridibundus</i>	BHGU
1774	32	Common/Arctic Tern	<i>Sterna hirundo/paradisaea</i>	CATE



**Fig. 2** Data example of the common goldeneye, the black-headed gull and the lesser black-backed gull, respectively

ENP depends on the sensor resolution of the camera and the focal length of the lens, and if the sensor resolution is fixed, ENP can be increased by choosing a long (in terms of focal length) telephoto lens. An additional advantage is that there is no need to feed classifiers with large (in terms of the number of pixels) images. For more details about the segmentation, see [9]. Examples of the original data set images are presented in Fig. 2. The first image in this figure illustrates that there can be more than one bird in the image. There are species in the test area that have a custom to fly in tight flocks, and in these cases, the result (in terms of data examples) is an image of several birds. Moreover, there might be more than just one bird species in these flocks. The custom of flying in tight flocks is an important feature in terms of identification for certain bird species [17]. As result of the segmentation, an image has only one bird left when there is a sparse flock of birds in the image. In the sparse flock case, the bird closest to the center of the image is retained, thus when a sparse flock has more than one bird species, the retained bird species is chosen randomly. In the tight flock case, the identification is based on the whole flock, and thus it is biased toward the most numerous bird species in the flock.

#### 4.1 Data Augmentation

Data augmentation is applied to the original data set. We have used our own method, in which the images are converted into various color temperatures according to step size,  $s$ . The lower and upper limit to the color temperature is 2000 K and 15000 K, respectively. For example, if  $s = 50$  (in K), the number of data examples of the augmented data set is  $(15000 - 2000)/50 + 1 * 24631 = 6453322$ . For more details, see [9]. In addition to the color conversion, the images are also rotated by a random angle between  $-20^\circ$  and  $20^\circ$  drawn from the uniform distribution. Motivation for this is that CNN is invariant to small translations, but not image rotation [18]. Number of images for each species (classes) in the augmented data set when  $s = 50$  and  $s = 200$ , respectively, are presented in Table 2. Figure 3 presents data examples of the output of the augmentation algorithm. The original image, fed into the data augmentation

**Table 2** Number of images for each class in the augmented data set

Class Label	# Original	# s = 50	# s = 200
WTEA	2450	641900	164150
SWSP	260	68120	17420
LOSP	396	103752	26532
GRCO	5612	1470344	376004
COEI	979	256498	65593
COGO	1164	304968	77988
VESC	236	61832	15812
RBME	263	68906	17621
GBBG	512	134144	34304
HEGU	4053	1061886	271551
LBBG	1481	388022	99227
COGU	3648	955776	244416
BHGU	1803	472386	120801
CATE	1774	464788	118858



**Fig. 3** Data example of the common eider. The image on the left is an augmented image with color temperature of 3800 K. The original image is in the middle with color temperature of 5800 K. The image on the right is an augmented image with color temperature of 7800 K

algorithm, has a color temperature of 5800 K, and the two augmented images have a color temperature of 3800 K and 7800 K, respectively.

## 4.2 Grouping Data

We trained our first classifier models on the entire data set, which was divided into the same number of classes as the data had species. However, there are more and less easily separable classes (assessed by human eye), and that led to an idea of grouping those species together that seem similar to human eye, thus our proposal in this respect is hierarchical [19]. In this approach, the number of classes decreases

on the top level of the classifier hierarchy, and thus resulting better separability of the data set. Classification inside the groups is dealt with the subsequent levels in cascades [20]. Figure 4 illustrates examples of clearly and weakly separable classes. The white-tailed eagle and the mute swan are examples of clearly separable classes, and the herring gull and the common gull are examples of weakly separable classes.

There are four groups on the top-level of the classification hierarchy. Two of these groups are actually single clearly separable species; *swans* (treated as single species here) and *white-tailed eagle*, respectively. *Gulls-and-terns* and *waterfowl* (including loons and common cormorant) form the other two groups, respectively. More groups are defined below the top level in order to get species level classification. Division of the classes into the groups are given in Table 3. The number of images for each group formed from the original data set and from two augmented data sets with  $s = 50$  and  $s = 200$ , respectively, are given in Table 4.



**Fig. 4** From left to right: the white-tailed eagle and the mute swan are examples of clearly separable classes. The herring gull and the common gull are examples of weakly separable classes

**Table 3** Division of the classes into groups

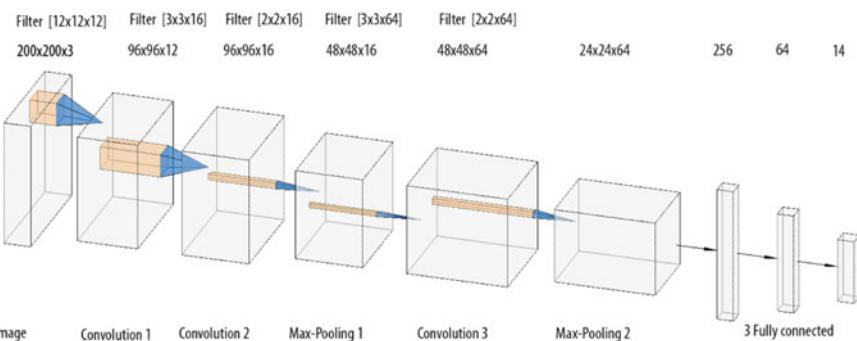
Class label	Top level	Second level	Third level	Fourth level
WTEA	White-tailed eagle	–	–	–
SWSP	Swans	–	–	–
LOSP	Waterfowl-1	Waterfowl-2	Waterfowl-2	–
GRCO	Waterfowl-1	Great cormorant	–	–
COEI	Waterfowl-1	Waterfowl-2	Waterfowl-2	–
COGO	Waterfowl-1	Waterfowl-2	Waterfowl-2	–
VESC	Waterfowl-1	Waterfowl-2	Waterfowl-2	–
RBME	Waterfowl-1	Waterfowl-2	Waterfowl-2	–
GBBG	Gulls-and-terns-1	Gulls-and-terns-2	Black-backed-gulls	GBBG
HEGU	Gulls-and-terns-1	Gray-backed-gulls	HEGU	–
LBBG	Gulls-and-terns-1	Gulls-and-terns-2	Black-backed-gulls	LBBG
COGU	Gulls-and-terns-1	Gray-backed-gulls	COGU	–
BHGU	Gulls-and-terns-1	Gulls-and-terns-2	Blackheaded-tern	BHGU
CATE	Gulls-and-terns-1	Gulls-and-terns-2	Blackheaded-tern	CATE

**Table 4** Number of images for each group

Group	# Original	# s = 50	# s = 200
White-tailed eagle	2450	641900	164150
Swans	260	68120	17420
Waterfowl-1	8387	2266300	579550
Waterfowl-2	2775	795956	203546
Gulls-and-terns-1	13271	3477002	889157
Gulls-and-terns-2	5570	1459340	373190
Gray-backed gulls	7701	2017662	515967
Blackheaded-tern	3577	937174	239659
Black-backed gulls	1993	522166	118858

## 5 Classification

All classifiers in this application share the same CNN model, which is shown in Fig. 5. Only the number of the neurons in the output changes according to the number of classes. This model has three convolution layers, each of which is followed by a rectified linear unit (ReLU) layer and the first two are followed by a cross-channel normalization layer (Local Response Normalization, LRN). The use of LRN is motivated by its ability to aid the generalization as its function may be seen as brightness normalization [2]. There are two max-pooling layers, the first is before the third convolution layer and the second is before the first fully connected layer. There is no max-pooling layer before the second convolution layer. The reason for this is the small ENP, and thus by omitting a max-pooling layer, all of the finest edges detected by the first convolution layer are transferred to the second convolution layer. The architecture is completed by three fully connected layers. The first two of them are followed by dropout layers, and each dropout layer is followed by ReLUs. The dropout was implemented by randomly setting the output neurons of the layer to zero with a probability of 0.5. The architecture is finally terminated by softmax



**Fig. 5** The CNN model for each classifier

**Table 5** Parameters for the convolution layers and the max-pooling layers of the CNN model

Layer	Filter	# Feature maps	Feature map size	Stride	Padding
Convolution 1	$12 \times 12$	12	$96 \times 96$	[2 2]	[1 1]
Convolution 2	$3 \times 3$	16	$96 \times 96$	[1 1]	[1 1]
Max-pooling 1	$2 \times 2$	16	$48 \times 48$	[2 2]	[0 0]
Convolution 3	$3 \times 3$	64	$48 \times 48$	[1 1]	[1 1]
Max-pooling 2	$2 \times 2$	64	$24 \times 24$	[2 2]	[0 0]

activation, which produces a distribution over the class labels with cross entropy loss function [21]. The input image is normalized and zero-centered before feeding it to the network. CNN with Mini-batch training and supervised mode as well as stochastic gradient descent with momentum is applied [21–24]. The L2 Regularization (weight decay) method for reducing over-fitting is also applied [21, 24, 25]. We kept the network size, in terms of free parameters, small due to limited capacity of computer resources. Thus, resulting in total of 92 feature maps which are extracted by convolution layers with kernel sizes  $[12 \times 12 \times 3] \times 12$ ,  $[3 \times 3 \times 12] \times 16$  and  $[3 \times 3 \times 16] \times 64$ , respectively. Total number of weights is about  $9.47 \times 10^6$ .

Images of a size of  $200 \times 200$  pixels are fed to each classifier. In the first convolution layer, this image size produces  $(200 - 12 + 2 * 1)/2 + 1 = 96$  square feature maps, i.e., there are  $96 \times 96 = 9216$  neurons in each feature map. Filter size, number of feature maps, feature map size in neurons, stride, and padding for each convolution layers and max-pool layers are given in Table 5. For each filter, Fig. 5 displays the number of feature maps as the triplet [a, b, c].

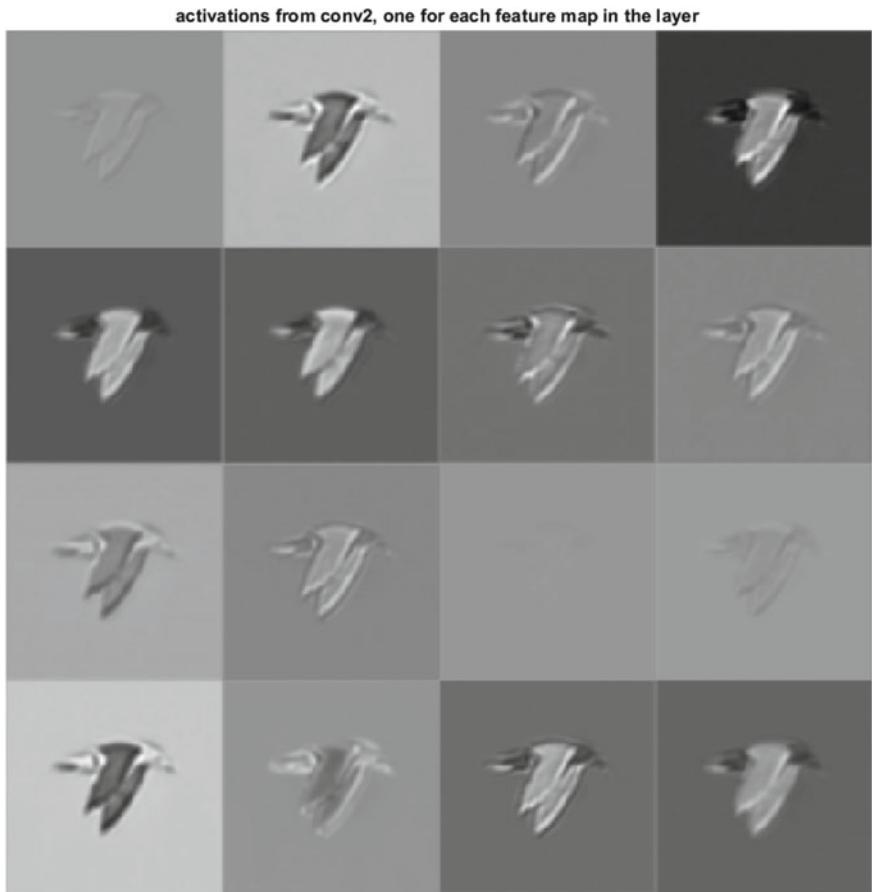
## 5.1 Hyperparameter Selection

We split the data set into a training set and a validation set as 70% and 30%, respectively. We used manual tuning for choosing the number of epochs. Initial weights for all layers are drawn from the Gaussian distribution with mean 0 and standard deviation 0.01. Initial biases are set to zero. The L2 value is set to 0.0005 and mini-batch size is set to 128.

## 5.2 Feature Extraction

The three convolution layers are designed to detect spatially distributed features from the training images. Usual disjunctive features are shape and general coloration of the bird. The ReLU (to introduce non-linearity) layer and the max-pooling (to increase spatial invariance) layer after the second and third convolution layers, respectively, may be seen as a refinement for the detected features due to the rectifying and down

sampling properties of these layers. Figures 6 and 7 illustrate the features, extracted by the CNN, for the classes LBBG and GBBG, respectively. These feature maps are from the second convolution layer. There is one frame for each 16 feature maps in the figure. These images are normalized, so that the minimum weight is 0 and the maximum is 1, i.e., the most negative weight has turned into zero (black). The mid-gray color (0.5) shows those areas in the image that have the minimum contribution to the features, and the most blackish or the most whitish areas denote maximum contribution to the features. The plain gray, or almost so, feature maps indicate that no significant features have been found in these maps. These feature maps show that the CNN is capable to give large weights on those areas of the bird plumage that are relevant for species identification. These areas are mainly: wing tips, feet, and a bill with these two pairs of gull species. As flying gulls usually have their feet concealed



**Fig. 6** Visualization of the feature extraction by the CNN model for the class LBBG. There are 16 feature maps in the figure extracted by the second convolution layer



**Fig. 7** Visualization of the feature extraction by the CNN model for the class GBBG. There are 16 feature maps in the figure extracted by the second convolution layer

by feathers, and their underside is not always visible in the images, the usage of this feature is minor. This leaves us the bill and the wing tip, and because the differences in the bill color and structure are only subtle, the most significant identification point is the wing tip. The great black-backed gull and the lesser black-backed gull also have a slight difference in the hue of their upper wing color, but this does not always seem to result in larger weights produced by the CNN for those areas, at least not large enough, because images of the great black-backed gull are even misclassified as the herring gull. Yet, the upper wing color is the key feature to distinguish between the *gray-backed gulls* and *black-backed gulls* [26].

**Table 6** Filter sizes for the second modified model of the original CNN model

Layer	Filter	# Feature maps	Feature map size	Stride	Padding
Convolution 1	$12 \times 12$	12	$96 \times 96$	[2 2]	[1 1]
Convolution 2	$7 \times 7$	16	$90 \times 90$	[1 1]	[0 0]
Convolution 3	$5 \times 5$	32	$86 \times 86$	[1 1]	[0 0]
Convolution 4	$4 \times 4$	64	$40 \times 40$	[1 1]	[0 0]
Convolution 5	$3 \times 3$	128	$18 \times 18$	[1 1]	[0 0]

### 5.3 Tests for Deeper CNN Model

It became clear during the development of this algorithm that the challenge, in terms of classification, lies in the group of *gulls-and-terns-1*, especially in the groups of *gray-backed gull* and *black-backed gull*. Considering the CNN model, the first option for a better performance should be a deeper model, i.e., more convolution layers. We modified the original model by adding the fourth convolution layer, followed by ReLU and max-pooling layers. This model had 128 filters with filter size of [5 × 5] in its fourth convolution layer. The first modified model was tested on the group of *black-backed gull*, but it failed to increase the performance of the original CNN model. Then we tested even deeper model by adding the fifth convolution layer, again followed by ReLU and max-pooling layers. In this case, the max-pooling layer before the third convolution layer in the original model was removed in order to have sufficient number of neurons left at the output of the architecture. We also modified the filter sizes of the second modified model. The modified filter sizes are given in Table 6. The two new max-pooling layers at the end of the second modified model have filter size of [2 2], respectively. When this model was tested on the group of *black-backed gull*, the result was the same as the first modified model, i.e., it did not achieve a better performance than the original CNN model in terms of true positive rate (TRP). Both test classifiers were trained on the augmented set, with  $s = 50$ , of only the images from the group of *black-backed gull*.

### 5.4 Dealing with Imbalanced Data

If we want to identify (classify) all the species that occur in the test area, we must accept that the training data set will be imbalanced, because there will be low numbers of training examples of the scarcest species. However, there are methods that can be used for imbalanced data set. Naturally, the first option would be to collect more data into the training data set, but this is not a very realistic option in our case. Resampling is a method that is easy to implement, and fast to run. This means that copies of data examples are added into the under-represented class, i.e., oversampling, or data examples are deleted from the over-represented class, i.e., undersampling [11]. However, we have augmented the original data set (resampling is

**Table 7** Imbalanced ratios of 13 classes to the class with the largest number of images (GRCO)

Class label	Ratio
WTEA	1:2
SWSP	1:22
LOSP	1:14
COEI	1:6
COGO	1:5
VESC	1:24
RBME	1:21
GBBG	1:11
HEGU	1:1
LBBG	1:4
COGU	1:2
BHGU	1:3
CATE	1:3

not used) with  $s = 50$ , and trained a reference classifier on the augmented data set. The results, in terms of performance, of the hybrid model (hierarchical and cascade model combined) trained on the grouped data set are compared with this reference classifier. The grouped data set is also augmented with  $s = 50$ , and both data sets are imbalanced. Class imbalance ratios (i.e., ratio of the number of images in a class to the class with the largest number of images) of the original data set for 13 classes, rounded to the nearest integer, are given in Table 7. The class with the largest number of images is GRCO, and it is omitted from the table. It can be seen from the table that there is severe imbalance between several classes and the class GRCO.

Another reference classifier is trained on a balanced data set. This data set is created by under-sampling method, so that the original data set is augmented with  $s = 50$ , and then  $236 \times 262 = 61832$  images are randomly chosen from each class, except for the class VESC, from which all of the images are chosen, because this class has the lowest number of data examples.

It is important to choose a suitable performance metric for classifiers trained on imbalanced data set. We have used confusion matrix as a tool to compare the classifiers. Precision (a measure for classifier exactness) and recall (a measure for classifier completeness, a.k.a. TPR) are metrics that have been calculated from confusion matrices. Receiver operating characteristic (ROC) curves and histograms of predictions are the tools that have been applied to determine thresholds for various classifiers trained on the grouped data set. Histograms present the predictions of a classifier fed by a test data set, which the classifier has never seen before. Thus, histogram shows the distribution of prediction of a classifier over a class by presenting the number of the predicted probabilities that falls into each bin. There are always only two classes in the histograms: the positive class (in red), and the negative class (in blue). If it is necessary to use histograms for more than two classes, then one of the classes is treated as the positive class, and the other classes are combined to

form the negative class. For all histograms, the x-axis is probability, and y-axis is the number of hits for each bin. Y-axis ranges from zero to the largest probability that hits a single bin in the histogram. The number of bins is always set to 10, and thus the bin width is 0.1.

## 5.5 Hybrid of Hierarchical and Cascade Model

In order to improve the performance of the classification algorithm compared to the single CNN classifier, we can use more than just one classifier, and we can divide the data set into suitable groups, which is done in Sect. 4.2. Eight classifiers have been trained on the grouped data sets. These classifiers form a hierarchy that is applied to classify the original data set. This architecture may also be seen as a hybrid between hierarchical and cascade models. The architecture is depicted in Fig. 8. The level of a classifier in the hierarchy, the data set (the groups) that it has been trained on, and the number of classes for each classifier are given in Table 8.

The first idea was merely to use cascade model of classifiers, so that the commonest species, determined by the distribution of the data set, would be filtered out (classified) at the first classifier. The second commonest species would be filtered out at the second classifier, and so forth. However, the early tests showed that there is no significant difference in performance, if those classes with better separability would be classified in a single classifier. Moreover, the cascaded approach would have led to a relatively large number of classifiers to be trained on, and thus increasing the training time. Nevertheless, the cascaded model was applied to some groups, and especially to the groups of the weakest separability, which are the gulls-and-terns groups. The prediction of a classifier for a test image is the vector-output of the softmax-layer, and it is given,

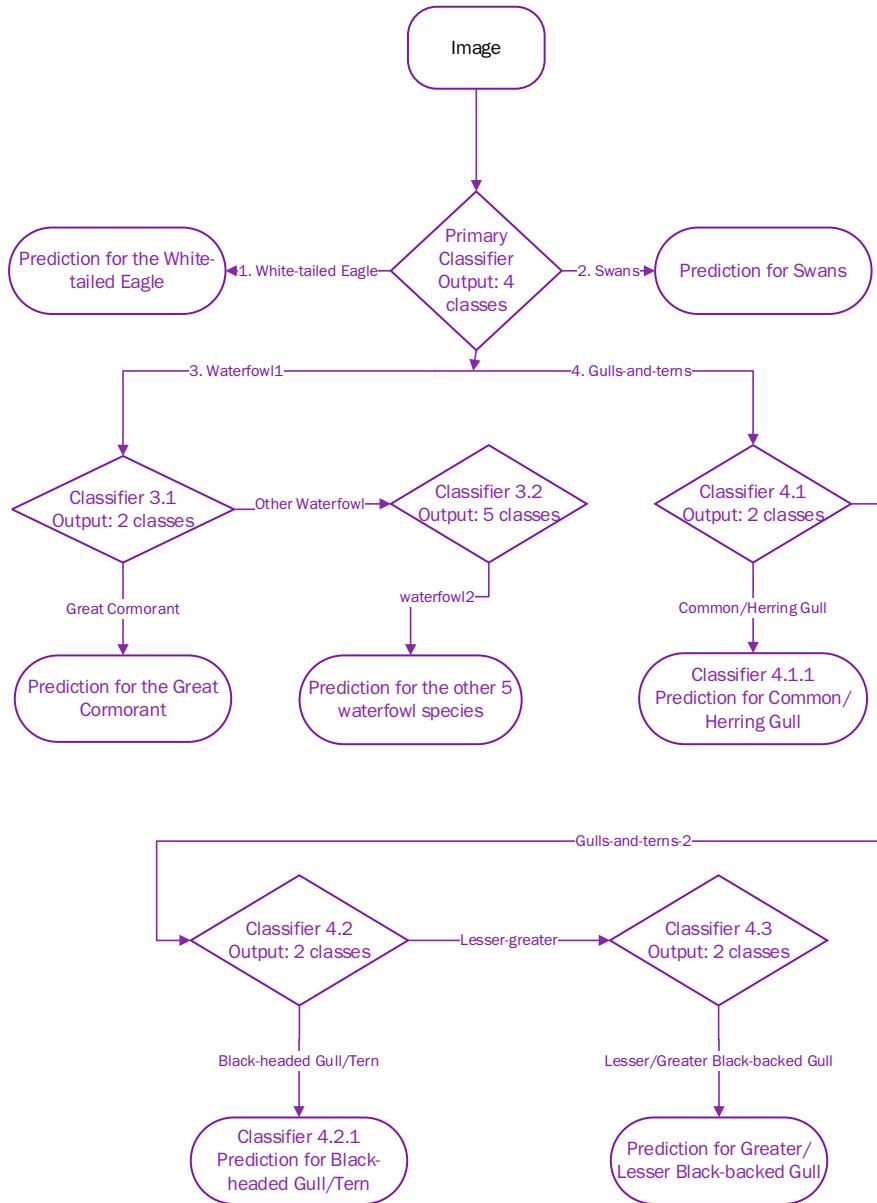
$$\mathbf{P} = [p_1, p_2, \dots, p_n], \quad (1)$$

where  $p_i$  is a probability for a *class<sub>i</sub>* as a result of the classification, and  $n$  is the number of classes. Classes are alphabetically ordered by their class labels. Thresholds are applied as follows:

$$c_i = \begin{cases} 1, & \text{if } p_i > \text{threshold}_i, \\ 0, & \text{otherwise,} \end{cases} \quad (2)$$

$$\mathbf{C} = [c_1, c_2, \dots, c_n], \quad (3)$$

where  $p_i$  is as in the  $\mathbf{P}$ -vector (1), and  $\text{threshold}_i$  is the threshold for  $\text{class}_i$ . As result of Eqs. (2) and (3) there will be exactly one element,  $c_i$ , turned to one in  $\mathbf{C}$ -vector, and the rest of the elements are turned to zeros. The class label is found according to the index of the element that is turned to one:



**Fig. 8** Hierarchy of classifiers

**Table 8** Classifiers trained for the hybrid model

Classifier	Level	Data set (the group)	# Classes
1	1	WTEA, SWSP, waterfowl-1, gulls-and-terns-1	4
2	2	Cormorant, waterfowl-2	2
3	2	Gray-backed-gulls, gulls-and-terns-2	2
4	3	Waterfowl-2	5
5	3	Blackheaded-tern, black-backed-gulls	2
6	3	HEGU, COGU	2
7	4	BHGU, CATE	2
8	4	LBBG, GBBG	2

$$j = \arg \max_j(\mathbf{C}), \quad (4)$$

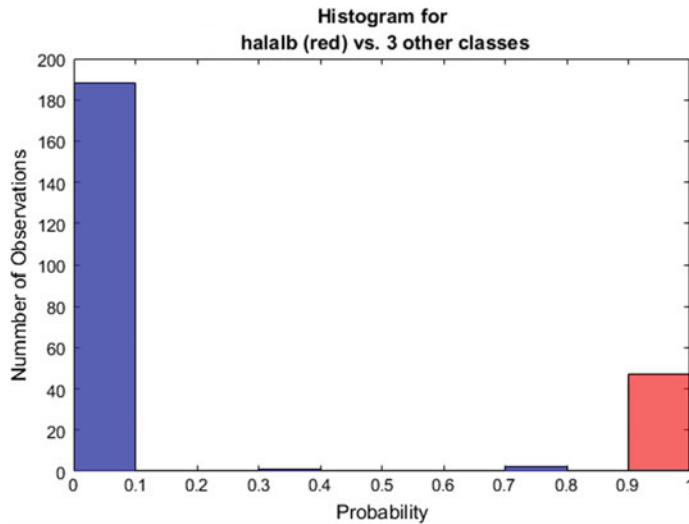
where  $j$  is the index of the predicted class.

## 5.6 Top-Level Classification

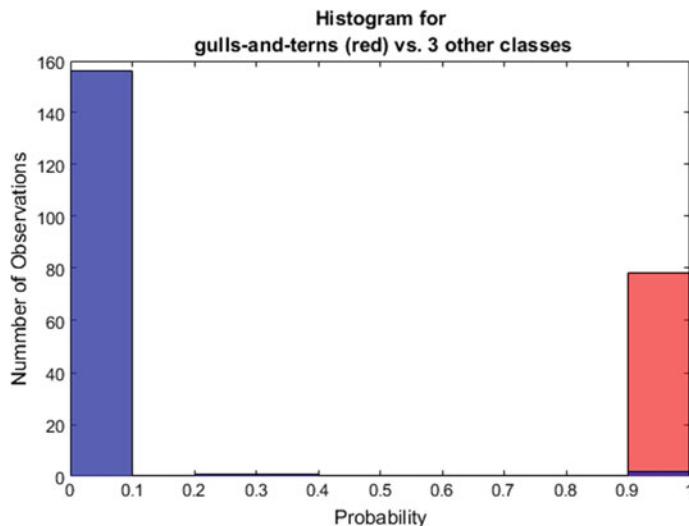
The top-level classifier is the most important in terms of TPR, because a possible misclassification will recur in subsequent hierarchy. This classifier deals with the groups: *swans*, *waterfowl-1*, *white-tailed eagle*, and *gulls-and-terns-1*. Class imbalance ratios of the top-level groups, rounded to the nearest integer, are given in Table 9. Considering the environmental license requirements, it is crucial to keep the number of false negative (FN, a data example from the positive class that is misclassified as the negative class) of the group of *white-tailed eagle* and the group of *gulls-and-terns-1* as low as possible, preferably at zero. Figures 9 and 10 illustrate the choice of possible threshold for the group of *white-tailed eagle*, and for the group of *gulls-and-terns-1*, respectively. These histograms are formed from the predictions of the top-level classifier (a.k.a. primary classifier), so that a histogram for the positive class and the negative class are plotted in the same graph, respectively. Equivalent ROC curves are computed based on the histograms, from which the TPRs and false positive rates (FPR) are calculated. ROC curves for the group *white-tailed eagle* and for the group *gulls-and-terns-1* are shown in Figs. 11 and 12, respectively. Both figures, the histogram and the ROC curve, for the group of *white-tailed eagle* show that this group is clearly separable, and thus it is easy to choose a suitable threshold for perfect

**Table 9** Imbalanced ratios of the top-level groups to the group with the largest number of images (*gulls-and-terns-1*)

Class label	Ratio
White-tailed eagle	1:5
Swans	1:51
Waterfowl-1	1:2



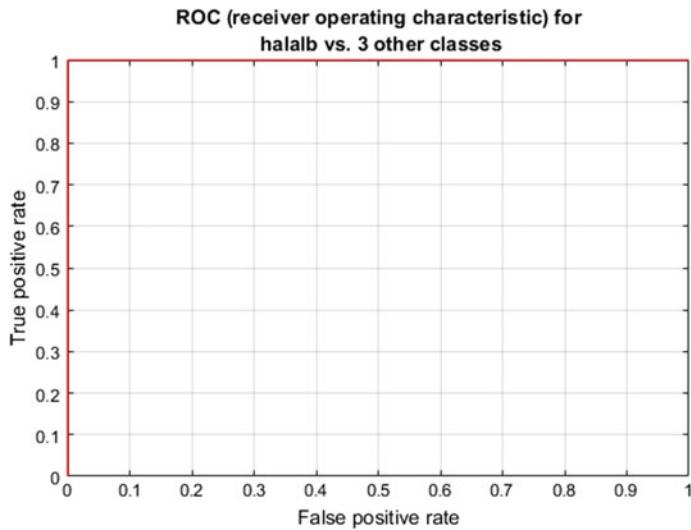
**Fig. 9** Histogram for the group *white-tailed eagle*



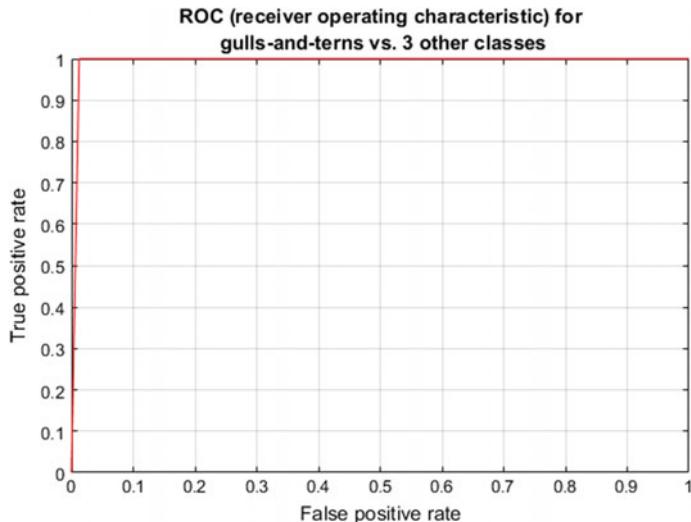
**Fig. 10** Histogram for the group *gulls-and-terns-1*

classification of the group. Generally, two values of probability can be read from the histogram, and use as a threshold: the lowest probability value of the positive class (LPPC), and the highest probability value of the negative class (HPNC).

In the case of the group of *white-tailed eagle*, these two probability values are not overlapped, and thus this class is clearly separable. The threshold can be set anywhere between 0.8 and 0.9 in order to classify this class perfectly. All true positives (TP,



**Fig. 11** ROC curve for the group *white-tailed eagle*



**Fig. 12** ROC curve for the group *gulls-and-terns-1*

a data example from the positive class that is correctly classified) will be classified correctly and there will be no false positives (FP, a data example from the negative class that is misclassified as the positive class), nor FNs. As the group of *white-tailed eagle* actually consists of only that single bird species, this also means that this classifier is capable to classify the white-tailed eagle in accordance with the environmental license.

In the case of the group of *gulls-and-terns-1*, the LPPC and the HPNC are overlapped. There are two data examples from the negative class that have the probability between 0.9 and 1, and all of the probabilities from the positive class fall into the same bin. Probabilities inside the bins cannot be read from the histograms, but the plotting software (MatLab) also prints the exact values for the probabilities. The LPPC and the HPNC for the group of *gulls-and-terns-1* is 0.9000 and 0.9643, respectively. If we choose 0.9 for the threshold, there will be two FPs, but if we choose 0.9643 for the threshold, there will be no FPs, nor FNs. In this case the number of FNs is the most important, because the lesser black-backed gull belongs to the group of *gulls-and-terns-1*, and it is particularly taken into account in the environmental license, so we cannot take the risk of misclassifying a gull at the top level of the hierarchy. Therefore, we must choose 0.9643 for the threshold. Table 10 shows the applied threshold for the top-level group. The threshold for the group of *white-tailed eagle* is set to 0.7415, because this is the exact value printed by the plotting software. One image of the great cormorant in the test data set is misclassified as the white-tailed eagle, and this causes the number of FPs to be one for the white-tailed eagle and the number of FNs to be one for the group *waterfowl-1*. This is acceptable error rate, because no white-tailed eagles are missed. Algorithm 1 describes the top-level classification process. This algorithm also defines a new pseudo-class, which means that this class does not exist in the data set, but it is used when the primary classifier fails to classify a test image correctly. Thus, it enables a definition of an unidentified bird (UNBI) class without explicitly including it in the real-world classes.

---

**Algorithm 1:** Classification on the top level.

---

```

Data: A test image
Result: Classification of the test image
image = zeroCentering(testImage);
TopLevelPrediction = classify(PrimaryClassifier, image);
if TopLevelPrediction > thresholdWhiteTailedEagle then
    | return WTEA; // the white-tailed eagle
else if TopLevelPrediction > thresholdSwans then
    | return SWSP; // swan species
else if TopLevelPrediction > thresholdGullsAndTerns then
    | // Classification for gull and tern species continues in Algorithm 3
else if TopLevelPrediction > thresholdWaterfowl then
    | // Classification for waterfowl species continues in Algorithm 2
else
    | return UNBI; // a pseudo-class for unidentified bird species
end

```

---

**Table 10** Thresholds for the top-level group

Group	Threshold	# FNs	# FPs
White-tailed eagle	0.7415	0	1
Swans	0.7000	0	0
Waterfowl-1	0.0083	1	0
Gulls-and-terns-1	0.9643	0	0

## 5.7 Classification of Waterfowl

Waterfowl are classified in the second level of the hierarchy, so that two classifiers are cascaded. The first one filters out the commonest class, GRCO, and all the other waterfowl are classified in the second classifier. Thresholds for the first classifier are given in Table 11. There is one FN, and accordingly, one FP as these thresholds are applied. The misclassified class is GRCO, which is the only class in the group of *cormorants*. Thresholds for the group of *waterfowl-2* is given in Table 12. All other classes have one FN, respectively, except for the class LOSP, which is clearly separable. Algorithm 2 shows the classification process for both waterfowl groups. Two new pseudo-classes are defined in the algorithm: unidentified waterfowl (UNWF), and unidentified small waterfowl (UNSW).

---

### Algorithm 2: Classification for the waterfowl groups.

---

```

Data: Image from the top level classifier in Algorithm 1, when TopLevelPrediction
      > thresholdWaterfowl
Result: Classification of the test image
predictionWF1 = classify(classifier_3.1, image);
if predictionWF1 > thresholdCormorant then
    | return GRCO; // the great cormorant
else if predictionWF1 > thresholdWF2 then
    | predictionWF2 = classify(classifier_3.2, image);
    | if predictionWF2 > thresholdLoons then
    |     | return LOSP; // loon species
    | else if predictionWF2 > thresholdGoldeneye then
    |     | return COGO; // the common goldeneye
    | else if predictionWF2 > thresholdEider then
    |     | return COEI; // the common eider
    | else if predictionWF2 > thresholdMerganser then
    |     | return RBME; // the red-breasted merganser
    | else if predictionWF2 > thresholdScooter then
    |     | return VESC; // the velvet scoter
    | else
    |     | return UNSW; // a pseudo-class for small unidentified waterfowl
    | end
else
    | return UNWF; // a pseudo-class for unidentified waterfowl
end

```

---

**Table 11** Thresholds for the group waterfowl-1

Class Label	Threshold	# FNs	# FPs
Cormorant	0.2627	1	0
Waterfowl-2	0.7000	0	1

**Table 12** Thresholds for the group waterfowl-2

Class label	Threshold	# FNs	# FPs
LOSP	0.0402	0	0
COEI	0.9909	1	0
COGO	0.0120	1	0
VESC	0.8810	1	0
RBME	0.9831	1	0

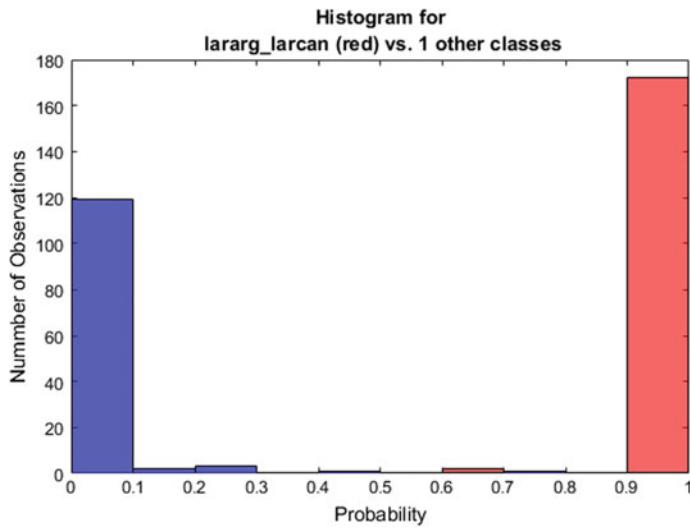
**Table 13** Number of images in the larger test data sets for gulls and terns

Pair of groups	# Positive class	# Negative class
{gray-backed-gulls, gulls-and-terns-2}	174	126
{blackheaded-tern, black-backed-gulls}	192	108
{HEGU, COGO}	92	82
{BHGU, CATE}	97	95
{LBBG, GBBG}	80	28

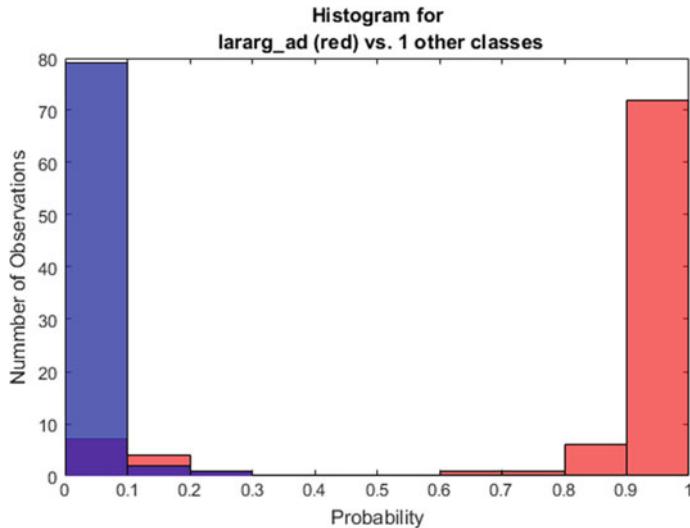
## 5.8 Classification of Gulls and Terns

Gulls and terns are classified in the second and third level of the hierarchy in cascade classifiers. We have used a larger test data set with more images for the groups of gulls-and-terns, which is enabled by the fact that the scarcest classes in the original data set are not included in these groups. In this way, we gain more robust threshold, though the original distribution is retained, and the test data set still has only images that the classifiers have never seen before. The number of images in these data set sets are given in Table 13. In this Table, the pair of groups or classes is in the first column from the left, so that the positive class is mentioned first. The following two columns are the number of images of the positive class and the negative class, respectively. We can calculate from the table that the class imbalance ratio for the most of the pairs of the fourth level of the hierarchy (the species level) is almost balanced. The pair {LBBG, GBBG} is the only significant exception having the class imbalanced ratio of 1:3, and because this pair also has the weakest separability, poor result for classification is expected in terms of TPR.

At the second level the commonest group, *gray-backed-gulls* is filtered out first from the group of *gulls-and-terns-2*, and then subsequently the group *blackheaded-tern*. Finally, the group *black-backed-gulls* is the only one left. Figure 13 shows the histogram for the group of *gray-backed-gulls*. It becomes clear from the histogram that the distributions of the positive class (*gray-backed-gulls*) and the negative class (*gulls-and-tern-2*) are overlapped, and that given we must make a choice for a suitable threshold while keeping in mind the terms of the environmental license. There are two choices for the threshold: 0.6000 with one FP, and 0.7590 with two FNs. We must choose 0.7590 even though it means weaker general performance for the hierarchical



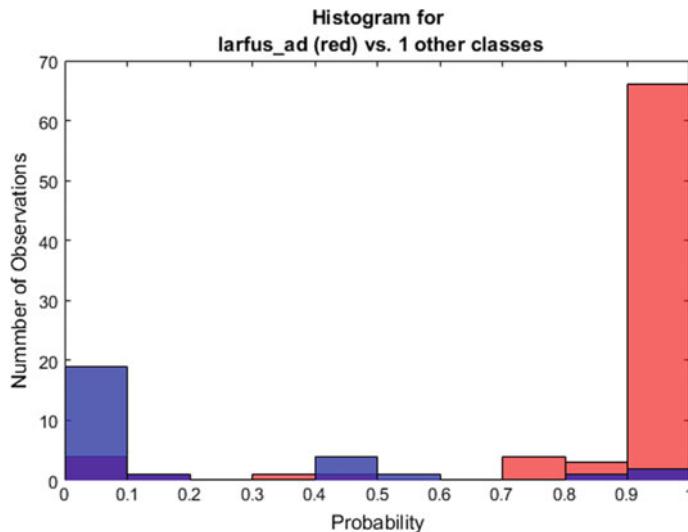
**Fig. 13** Histogram for the group *gray-backed-gulls*



**Fig. 14** Histogram for the class HEGU (herring gull)

classifier. This is because we do not want any member of the class LBBG misclassified on the second level. As result of applying this threshold, there will be two images of the group *gray-backed-gulls* misclassified as *gulls-and-terns-2*.

Species level classification is reached on the fourth (third for gray-backed gulls) level of the hierarchical classifiers. This includes pairs of classes with the weakest separability: {HEGU, COGU}, and {LBBG, GBBG}. The overlap of the distributions of the positive and the negative classes are illustrated in Figs. 14 and 15. The



**Fig. 15** Histogram for the class LBBG (lesser black-backed gull)

class HEGU is the positive class in Fig. 14, and the class COGU is the negative class. The class LBBG is the positive class in Fig. 15, and the class GBBG is the negative class. The best value for a threshold for separating the HEGU from COGU, in terms of classifier performance, is 0.2134. This means zero FP, but eleven FNs, i.e., eleven images of herring gulls will be misclassified as common gulls. Classification of the pair {BHGU, CATE} is straightforward owing to the fact that with the chosen threshold it has the number of FNs and FPs equal to zero. See Table 14 for thresholds for the groups of gulls and terns. The best option for a threshold of the class LBBG is 0.9993 when the number of FNs is 12. Algorithm 3 shows the classification process for the group of *gulls-and-terns-1*. Five new pseudo-classes are defined in this algorithm: gray-backed gull (GBGU, either the herring gull or the common gull), black-headed (BHTE, either the black-headed gull or tern species), black-backed gull (BBGU, either the lesser black-backed or the great black-backed gull), non-gray-backed-gull (NGGU, either BHTE or BBGU), and unidentified gull or tern (UNGU).

**Table 14** Thresholds applied to the pair of groups of gull and tern species

Class label	Threshold	# FNs	# FPs
{gray-backed-gulls, gulls-and-terns-2}	0.7590	2	0
{blackheaded-tern, black-backed-gulls}	0.2524	0	0
{HEGU, COGO}	0.2134	11	0
{BHGU, CATE}	0.8124	0	0
{LBBG, GBBG}	0.9993	12	0

## 6 Results

Results for comparing the classifiers are viewed through generalization. The hybrid of hierarchical and cascaded model achieved average performance of 0.9460 (TPR). The reference classifiers have average TPRs as follows: for the imbalanced reference classifier (IMBRC), 0.8195 and for the balanced reference classifier (BRC), 0.8307, respectively. The total number of misclassification for the hybrid model was 16. This number for the reference classifiers was 45 for the IMBRC, and 71 for the BRC. Average precision for the hybrid model was 0.9619. Average precision for the IMBRC was 0.8809, and for the BRC 0.7919. TPRs for the top-level groups and the class LBBG are given in Table 15. The reference classifiers were trained on ungrouped classes, therefore the numbers for TPRs of the groups have been averaged of the numbers of those classes that the groups consist of.

Confusion matrix for the top-level groups is given in Table 16. This confusion matrix also includes the pseudo-class UNBI. Naturally, the number of TPs are zero for pseudo-classes, because these classes are only defined for failure of the classifiers. Confusion matrix for the classes are given in two parts, because it is too big to fit in the page. Table 17 presents the first part of the confusion matrix including the group of *waterfowl-1*, i.e., the classes: LOSP, GRCO, COEI, COGO, VESC, and RBME. This confusion matrix also includes the pseudo-classes: UNSW, and UNWF. One test image of GRCO is presented in the top-level confusion matrix, therefore the number of test images for the class GRCO is 99 in the waterfowl confusion matrix. Table 18 presents the second part of the confusion matrix including the group *gulls-and-terns-1* (the classes: GBBG, HEGU, LBBG, COGU, BHGU, and CATE). The five pseudo-classes defined in Algorithm 3 for gulls and terns are omitted in order to save space, and because no image of any of the subgroups of the *gulls-and-terns-1* was misclassified as any of these pseudo-classes.

**Table 15** TPRs for the hybrid model and the reference classifiers

Classifier	WTEA	SWSP	Waterfowl-1	Gulls-and-terns-1	LBBG
Hybrid	1	1	0.9935	1	0.9231
Imbalanced reference	0.9773	0.4000	0.7629	0.7691	0.6923
Balanced reference	1	0.8000	0.8621	0.7762	0.8846

**Table 16** Confusion matrix for the top-level groups of the hierarchy

	WTEA	SWSP	Waterfowl-1	Gulls-and-tern-1	UNBI
WTEA	44	0	0	0	0
SWSP	0	5	0	0	0
Waterfowl-1	1	0	153	0	0
Gulls-and-terns-1	0	0	0	236	0
UNBI	0	0	0	0	0

---

**Algorithm 3:** Classification for the group gulls-and-terns.

---

```

Data: Image from the top level classifier in Algorithm 1, when TopLevelPrediction
      > thresholdGullsAndTerns

Result: Classification of the test image
predictionGullsTerns = classify(classifier_4.1, image);
if predictionGullsTerns > thresholdGrayGulls then
    predictionHerringCommon = classify(classifier_4.1.1, image);
    if predictionHerringCommon > thresholdHerring then
        | return HEGU; // the herring gull
    else if predictionHerringCommon > thresholdCommon then
        | return COGU; // the common gull
    else
        | return GBGU; // a pseudo-class for a 'gray-backed gull' i.e.,
                      | either the herring gull or the common gull
    end
else if predictionGullsTerns > thresholdGullsTerns2 then
    predictionGullsTerns2 = classify(classifier_4.2, image);
    if predictionGullsTerns2 > thresholdBlackHeaded then
        predictionBlackHeaded = classify(classifier_4.2.1, image);
        if predictionBlackHeaded > thresholdBHGU then
            | return BHGU; // the black-headed gull
        else if predictionBlackHeaded > thresholdCATE then
            | return CATE; // the common/arctic tern
        else
            | return BHTE; // a pseudo-class for either the black-headed
                          | gull or tern species
        end
    else if predictionGullsTerns2 > thresholdBlackBacked then
        predictionBlackBacked= classify(classifier_4.3, image);
        if predictionBlackHeaded > thresholdLBBG then
            | return LBBG; // the lesser black-backed gull
        else if predictionBlackHeaded > thresholdGBBG then
            | return GBBG; // the great black-backed gull
        else
            | return BBGU; // a pseudo-class for either the lesser
                          | black-backed or the great black-backed gull
        end
    else
        | return NGGU; // a pseudo-class for 'non-gray-backed gull'
    end
else
    | return UNGU; // a pseudo-class for unidentified gull or tern species
end

```

---

As the number of images in the test data set is 439, we must split this number between the three confusion matrices. The first confusion matrix is for all 439 images, but because the classes WTEG and SWSP are only presented in this confusion matrix, the sum of the number of test images in the other two confusion matrices is  $439 - 50 = 389$ . The second confusion matrix presents results for 153 test images and the third for 236 test images, so that the sum is  $153 + 236 = 389$  test images.

Confusion matrix for the IMBRC is given in two tables: Tables 19, and 20, respectively. The class WTEA is included in both tables, because there are FPs and/or FNs for it in the two tables. However, the number of TPs for the class WTEA is only

**Table 17** Confusion matrix for the classes of the group *waterfowl-1*

	LOSP	GRCO	COEI	COGO	VESC	RBME	UNSW	UNWF
LOSP	7	0	0	0	0	0	0	0
GRCO	0	98	0	1	0	0	0	0
COEI	0	0	16	0	0	0	1	0
COGO	0	0	0	19	0	1	1	0
VESC	0	0	0	0	3	0	1	0
RBME	0	0	0	0	0	4	1	0
UNSW	0	0	0	0	0	0	0	0
UNWF	0	0	0	0	0	0	0	0

**Table 18** Confusion matrix for the classes of the group *gulls-and-terns-1*

	GBBG	HEGU	LBBG	COGU	BHGU	CATE
GBBG	4	1	4	0	0	0
HEGU	0	66	0	6	0	0
LBBG	0	0	26	0	0	0
COGU	0	1	0	64	0	0
BHGU	0	0	0	0	32	0
CATE	0	0	0	1	0	31

**Table 19** Confusion matrix for the imbalanced reference classifier, part one

	WTEA	LOSP	GRCO	COEI	COGO	VESC	RBME
WTEA	43	0	0	0	0	0	0
LOSP	0	6	0	0	1	0	0
GRCO	2	0	97	0	1	0	0
COEI	0	0	0	17	0	0	0
COGO	1	0	0	1	19	0	0
VESC	0	0	0	1	0	3	0
RBME	0	0	0	0	0	0	5

given in the first table. The same test data set, as with the hybrid model, has been used when the reference classifiers were tested. The total number of images, 439, is again divided into two tables. The first table covers 197 test images and the second table covers 242 test images.

Confusion matrix for the BRC is also given in two tables: Tables 21 and 22, respectively. There are no FPs or FNs for the class of WTEA in the second confusion matrix, so the class can be omitted from this table.

The results for the modified CNN models compared to the original CNN model are given in Table 23. All three models were trained on the same augmented data

**Table 20** Confusion matrix for the imbalanced reference classifier, part two

	WTEA	SWSP	GBBG	HEGU	LBBG	COGU	BHGU	CATE
WTEA	—	0	0	0	0	1	0	0
SWSP	0	2	0	2	0	1	0	0
GBBG	1	0	3	2	2	1	0	0
HEGU	1	0	0	67	0	4	0	0
LBBG	0	0	4	2	18	1	1	0
COGU	0	1	0	8	0	56	0	0
BHGU	0	0	0	0	1	0	30	1
CATE	0	0	0	1	0	1	3	27

**Table 21** Confusion matrix for the balanced reference classifier, part one

	WTEA	LOSP	GRCO	COEI	COGO	VESC	RBME
WTEA	44	0	0	0	0	0	0
LOSP	0	6	0	0	1	0	0
GRCO	4	1	91	1	1	2	0
COEI	0	0	0	16	0	1	0
COGO	0	0	2	1	15	2	1
VESC	0	0	0	1	0	3	0
RBME	0	0	0	0	0	0	5

**Table 22** Confusion matrix for the balanced reference classifier, part two

	SWSP	GBBG	HEGU	LBBG	COGU	BHGU	CATE
SWSP	4	0	0	0	1	0	0
GBBG	0	5	0	4	0	0	0
HEGU	1	0	48	4	18	0	1
LBBG	0	2	0	23	0	1	0
COGU	0	1	10	1	52	1	0
BHGU	0	0	0	1	2	28	1
CATE	0	0	1	0	3	0	28

**Table 23** TPRs for the modified CNN models compared to the original CNN model

Model	Training	Generalization
Modified 1	0.9977	0.8384
Modified 2	0.9827	0.7464
Original	0.9989	0.8597

set, which only consisted of the images from the group of *black-backed gull*. These models were tested as single classifiers. There are TPRs for both training and generalization (tested on images that the classifier has never seen before) in the Table. The first modified model had four convolution layers, and the second had five convolution layers. The models were tested only on the group of *black-backed gull* in these tests.

## 7 Discussion

The tests showed that the hybrid model is significantly better, in terms of performance, than the reference classifiers. The only problematic class, in terms of the environmental license, is the LBBG. Even though it had the number of FNs zero in the test for the hybrid model, the number of FNs was 12 in the test for the gulls and terns only. The number of test images in the latter test was larger, and this gives insight into real-world implementation. The number of possible FPs is not significant in this context, because it would just mean that other gull species, more likely great black-backed gulls, are misclassified as LBBG. Therefore, it is advisable to combine the classes LBBG and GBBG into a single class, i.e., not classify the group *black-backed-gulls* any further.

The BRC performed better than the IMBRC, in terms of TPR. However, the number of misclassification is 71 for the BRC and 45 for the IMBRC. The difference is explained by the better average precision of the IMBRC. Precision increases as the number of FPs decreases, and TPR increases as the number of FNs decreases. This means that TPR is more significant metric than precision in our context, and thus the BRC would be the second choice after the hybrid model. The IMBRC showed poor performance even though it was trained on larger data set ( $6.45 * 10^6$  versus  $8.66 * 10^5$ ) than the BRC. This implies that straightforward use of a single classifier on an imbalanced data set gives poor performance in terms of TPR. This result is based on relatively low number of data examples, which is often the case in real-world application, but this method could perform better when trained on significantly larger training data set. However, if precision is an important criterion, then this method may be considered for real-world usage.

The top-level group has the number of FPs equal to one in its confusion matrix (Table 16). This FP is a misclassified GRCO as WTEA. This is, of course, a FN for the class GRCO. However, this is acceptable as no WTEA is misclassified, and thus the number of FNs for the class WTEA is zero. The group *waterfowl-1* also shows good results, and there are only five misclassification. It seems that grouping the original classes is useful approach to this kind of real-world classification problem. By grouping, you can confine the most difficult classification problem to the one group or even just to one subgroup. This approach indicates where the challenge lies. In this context the challenge are the groups of *gray-backed-gulls* and *black-backed-gulls*, respectively. The bird species that these groups consist of are very similar in terms of morphology. This leads to a conclusion (assessed by human eye) that the overlapped area of the classification boundary is clearly wide for both groups.

This suggests that significant increase of classification performance can be achieved only by collecting more images of these groups.

Surprisingly, the modified CNN models with architecture of more than three convolution layers, did not perform better than the original CNN model. This implies that the original model, with the architecture of three convolution layers, is capable to extract all relevant features from the training images, and additional convolution layers cannot adduce any more information.

The measured performance of the image classification has been obtained without using the parameters supplied by the radar. Those parameters, especially the speed of an object, provide additional and relevant a priori knowledge to the system. It is measured by the radar system that there are significant differences in flight speed between the groups of *waterfowl-1* and *gulls-and-terns-1*, and this can be utilized to turn a misclassification into the correct one.

## References

1. Li, F., Li, S., Zhu, C., Lan, X., Chang, H.: Class-imbalance aware CNN extension for high resolution aerial image based vehicle localization and categorization. In: 2017 2nd International Conference on Image, Vision and Computing (ICIVC), Chengdu (2017)
2. Krizhevsky, A., Sutskever, I., Hinton, G. E.: Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 84–90 (2017)
3. Mao, R., Lin, Q., Allebach, J.: Robust convolutional neural network cascade for facial landmark localization exploiting training data augmentation. In: Imaging and Multimedia Analytics in a Web and Mobile World 2018, pp. 374-1-374-5(5) (2018)
4. Jia, S., Wang, P., Jia, P., Hu, S.: Research on data augmentation for image classification based on convolution neural networks. In: 2017 Chinese Automation Congress (CAC), Jinan (2017)
5. Li, H., Lin, Z., Shen, X., Brandt, J., Hua, G.: A convolutional neural network cascade for face detection. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston (2015)
6. Rachmadi, R., Uchimura, K., Koutagi, G., Komokata, Y.: Japan road sign classification using cascade convolutional neural network. In: ITS (Intelligent Transport System) World Congress, Tokyo, pp. 1–12 (2016)
7. Robin radar models. In: Robin Radar Systems B.V. (Accessed 2019). <https://www.robinradar.com/>
8. pelco-D protocol. In: Bruxy REGNET. (Accessed 2019). <http://bruxy.regnet.cz/programming/rs485/pelco-d.pdf>
9. Niemi, J., Tanttu, J.: Automatic bird identification for offshore wind farms. In: Bispo, R., Bernardino, J.C., Costa, J.L., (Eds.), Wind Energy and Wildlife Impacts, Cham, pp. 135–151 (2019)
10. Mirzaei, G., Jamali, M., Ross, J., Gorsevski, P., Bingman, V.: Data fusion of acoustics, infrared, and marine radar for avian study. *IEEE Sensors J.* **15**(11) (2016)
11. Batista, G., Prati, R., Monard, M.: A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explor. Newsl.*, 20–29 (2004)
12. Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. *Comput. Learn. Theory* **904**, 23–37 (1995)
13. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, pp. 511–518 (2001)

14. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, pp. 886–893 (2005)
15. Yoshihashi, R., Kawakami, R., Iida, M., Naemura, T.: Evaluation of bird detection using time-lapse images around a wind farm. *Wind Energy* **20**(12), 1983–1995 (2017)
16. Malling Olsen, K., Larsson, H.: *Terns of Europe and North America*. Helm, London (1995)
17. Madge, S., Burn, H.: *Wildfowl, an Identification Guide to the Ducks, Geese and Swans of the World*. Helm, London (1988)
18. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition. In: International Conference on Computer Vision, Kyoto, pp. 2146–2153 (2009)
19. Silla, C., Freitas, A.: A survey of hierarchical classification across different application domains. *Data Min Knowl Disc* **22**(31) (2011)
20. Sun, Y., Wang, X., Tang, X.: Deep convolutional network cascade for facial point detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3476–3483 (2013)
21. Bishop, C.M.: *Pattern Recognition and Machine Learning*. Springer, New York (2006)
22. Y.L., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE 86, 11, New York, pp. 2278–2324 (1998)
23. Li, M., Zhang, T., Chen, Y., Smola, A. J.: Efficient mini-batch training for stochastic optimization. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge, New York, pp. 661–670 (2014)
24. Murphy, K.P.: *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, USA (2012)
25. Haykin, S.: *Neural Networks: A Comprehensive Foundation*, 2nd edn. Prentice Hall/Pearson, New York (1994)
26. Malling Olsen, K., Larsson, H.: *Gulls of Europe. Asia and North America*. Helm, London (2003)

# Deep Learning for Person Re-identification in Surveillance Videos



**Swathi Jamjala Narayanan, Boominathan Perumal, Sangeetha Saman and Aditya Pratap Singh**

**Abstract** In the recent years, Closed Circuit Television (CCTV) is viewed as the basis for providing security. One of the most important aspects of CCTV surveillance systems security mechanism is to re-identify a person captured in one of the camera across different surveillance cameras. Re-identification has a major role in several applications like automated surveillance of universities, offices, malls, home and restricted environments like embassies or laboratories with strong security restrictions. Traditionally, identifying a person in a video was practiced under the set of same external conditions (like same illumination, viewpoint, back ground conditions etc.). But when it comes to automated re-identification in a CCTV surveillance system, several challenges emerge as the environment is uncontrolled and keeps varying, further the poses of the person and the angles of the cameras capturing the videos also incur additional challenge for the task considered. When a person disappears from one camera view for a period of time, he should be recognized in another view of camera at a different location when there are environmental disturbances like variation in illumination, crowded scene, partial occlusions, physical appearance variations, full occlusions, view point variations, background clutter, shadows and reflections, etc. In this chapter, the major focus is on the techniques of deep learning used to develop an end-to-end re-identification system highlighting the methods to handle the uncontrolled environment challenges mentioned. An end-to-end re-identification task consists of sequence of steps namely pedestrian detection, person tracking followed by person re-identification. Given a video sequence or an image as an input, firstly the humans are detected from the video sequence as a process of pedestrian detection. The person tracking within the camera is conducted, to find the different

---

S. J. Narayanan (✉) · B. Perumal · S. Saman · A. P. Singh  
School of Computer Science and Engineering, VIT, Vellore 632014, India  
e-mail: [swathi.jns@gmail.com](mailto:swathi.jns@gmail.com)

B. Perumal  
e-mail: [boomi051281@gmail.com](mailto:boomi051281@gmail.com)

S. Saman  
e-mail: [sangeethacse1990@gmail.com](mailto:sangeethacse1990@gmail.com)

A. P. Singh  
e-mail: [adi18jan1999@gmail.com](mailto:adi18jan1999@gmail.com)

poses of the probe if needed. Then the re-identification process is conducted where the deep learning models are used to re-identify the person with the help of gallery set of videos and evaluates the similarities of gallery set and the person of interest by using deep learning metrics. The re-identification results end as a retrieval process where all similar images of the person of interest are retrieved. Several benchmark datasets considered in literature for re-identification system are VIPeR, ETHZ, PRID, CAVIAR, CUHK01, CUHK02, CUHK03, i-LIDS, RAiD, MARS, etc.

**Keywords** Deep learning · Re-identification · Video surveillance

## 1 Introduction

The most important aspect of any intelligent closed-circuit television (CCTV) surveillance system is to accomplish the task of re-identification of humans which is popularly termed as Person Re-identification [1, 2]. The objective of such system is to find out whether a person showing up in one camera is coming again in another camera i.e. to determine whether a pair of humans appearing in various cameras with non-overlapping views [3] has the same identity or different identity. Engaging or hiring human operators to track the person of interest would be highly time-consuming process as they need to spend more time and most of the time it ends as an exhaustive task for the operators. To overcome this situation, automated computer vision system with less human intervention is more suitable to assist the human operators in identifying a person over a set of non-overlapping or disjoint cameras. The advent of conducting research work in this area is to increase demand of public safety with the help of widespread large camera networks placed in and around the public places like theme parks, shopping malls, universities, airports, etc. To achieve the above-mentioned goals, it is very costly to depend entirely on human workers in order to accurately recognize or track a person of interest across several cameras.

In early days, the person re-identification was considered as a multi-camera tracking problem where appearance based models were used with the geometry calibration with disjoint cameras in the surrounding environment. In the year 2005, the term re-identification was coined by Zajdel et al. [4] from the university of Amsterdam where they tried to re-identify a person when he departs from the camera view and appears once again [4]. In the year 2006, Gheisasi et al. [5] applied spatio-temporal segmentation algorithm and used visual signs of the persons as input a for foreground detection. The problem was solved as image-based Re-id rather than as video based one. This was the first work representing the isolating the person Re-id from multi-camera tracking. Henceforth, the problem of Re-is is considered as an independent computer vision task. Further in the year 2010, there were two major works which proved that using multiple frames per person would effectively improve over the single frame version [6]. In the year 2014, Yi et al. [7] and Li et al. [8] employed Siamese neural network successfully for person re-identification problem where a

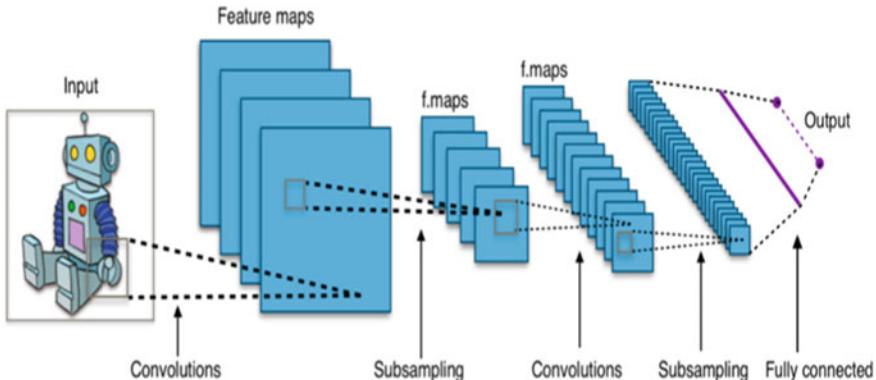
pair of input images where correctly determined with same identity using the network. This network helped to overcome the major issue of re-id problem which was having a lack in number of training samples. In the same year in 2014, Xu et al. proposed an end-to-end image-based re-identification model [9] where they combined the concept of detection and re-identification scores. The detection was used to find the commonness and re-identification was performed to find the uniqueness of the persons. The problem of person re-identification can be solved using any of the two systems namely handcrafted systems and deep learning systems. Re-identification system includes two components namely pedestrian detection and distance metrics. In handcrafted systems the features are extracted and passed on to re-identification system where as in deep learning systems, learning the features is inherent in the deep learning architecture and provides improved results compared to hand crafted systems. This chapter focuses on the preliminaries of deep learning algorithms followed by re-identification datasets and the different architectures, activation functions, loss functions and evaluation protocols used in re-identification application.

## 2 Preliminaries of Deep Neural Networks

This section discusses in brief on the basic deep learning models used in computer vision task. The models discussed are Convolutional Neural Network, LeNet5, AlexNet, ZFNet, VGGNet, GoogleLeNet, ResNet, Recurrent Neural Network, Siamese Neural Network. All these networks are based on CNN as a basic model and they vary in their architecture with respect to number of hidden layers, activation functions, loss functions and training mechanism.

### Convolutional Neural Network

In the domain of deep learning, most of the works on Convolutional Neural Networks (CNN) were performed to analyze visual images [10]. The network limits the process of including preprocessing step since the network learns the features automatically using filters and hence this avoids the feature design process. The convolution neural network is composed of three main layers having input, output and multiple hidden layers. The hidden layer further consists of convolution layers, activation functions, pooling layers, fully connected layers and normalization layers. The convolution layer employs the convolution operation on the input and forwards the result to the subsequent layer, where each neuron handles the data only for its receptive field. This avoids using a greater number of weights and allows the network to be deeper with less parameter. The activations functions commonly used in CNN are ReLU, Tanh and Softmax activation function. The pooling layers aim to continuously decrease the spatial size of the features in order to help in reducing the number of parameters and computations in the network. The pooling layer operates on each feature map independently. The commonly used pooling operations are max pooling and average pooling. The fully connected layer is to connect each neuron in one layer to every neuron in another layer. The receptive field (input area of the neuron) of each layer



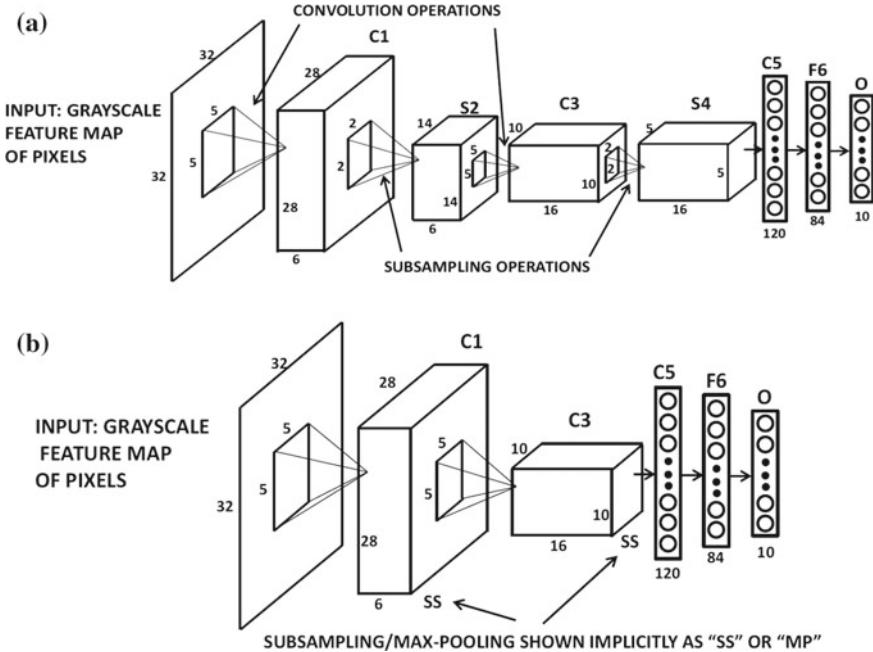
**Fig. 1** Basic CNN architecture

varies. As in fully connected layer, the convolution layer doesn't take input from every element of the prior layer, besides it takes input from a controlled subarea of the previous layer. Having weights and bias in each neuron, these weight vectors and bias vectors form a filter representing some feature of the input. The main strength of the CNN is that lots of neurons share the same filter thus eliminating the memory track of each receptive field taking up their corresponding bias and weight vectors. The other distinguishing feature of CNN is that it has 3D volume of data in terms of width, height and depth. Second feature is that it has layers of different types connected locally and completely and further stacked to build the CNN architecture. The architecture guarantees that the trained filters generate results to a spatially local input pattern and as the layers increase and get stacked up it would lead to nonlinear filters that become gradually global. The third distinct feature is on shared weights, i.e., each filter getting duplicated across the layers. The basic CNN architecture is given in Fig. 1.

### LeNet-5

**LeNet-5** is a kind of convolutional network which was mainly intended for performing handwritten and machine-printed character recognition [11]. The network has a total of 7 layers which includes two convolutional layers, two pooling layers, two fully connected layers and one output layer. LeNet-5 uses  $5 \times 5$  kernels of stride 1 and  $2 \times 2$  subsampling of stride 2. It is considered as the base model for various other successful deep CNN architectures. Figure 2 represents the LeNet-5 architecture where Fig. 2a demonstrates the architecture with subsampling or max-pooling layers whereas it is not a major focus of representation in other architectures like AlexNet. The same is represented in Fig. 2b. In the current architectural representations, max pooling is replaced in place of subsampling layer and they also occur less frequently than convolution layers.

LeNet-5 is vastly narrow in accommodating the recent standards. The architecture retains the basic principles and the most commonly used activation function is the sigmoid activation function. It accommodates RBF units in the final layer having the



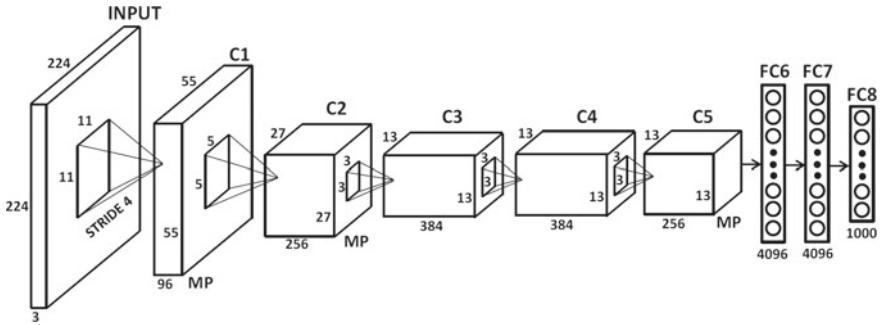
**Fig. 2** **a** Detailed architectural representation of LeNet-5 [12], **b** LeNet-5 brief representation [12]

prototype of every unit relating to the input vector and the output produced is the Squared Euclidean distance between them. In recent standards, the practice of using RBF is avoided and instead softmax units with log-likelihood loss on multinomial label outputs are used. The major applications of LeNet-5 is on character recognitions and is widely used to read the characters in bank cheques.

### AlexNet

**AlexNet** is a 8 layered CNN architecture that won ImageNet challenge 2012 and produced the widespread popularity for CNN architectures in the area of computer vision [13] Fig. 3 demonstrates the AlexNet architecture.

In Fig. 3, each convolution layer follows ReLu activation functions which is not explicitly shown and the max pooling layers denoted as MP follow only subset of convolution-ReLU combination layers .The architecture is composed of 5 convolutional layers and 3 fully connected layers. The first convolution layer comprises of 96 filters of size  $11 \times 11$  at stride 4 and second convolution layer consists of 256 filters of size  $5 \times 5$ . Third, fourth, and fifth convolution layers consists of 384, 384, and 256 filters respectively of size  $3 \times 3$  at stride 1. First, second, and final FC layers consists of 4096, 4096, and 1000 neurons. The most significant characteristic of the AlexNet is the use of non-linear activation function (ReLU) and it also uses heavy data augmentation. The role of ReLU activation function towards increasing the training speed of CNN is firstly exhibited in AlexNet architecture. This proved ReLU is far better than steeping activation functions like sigmoid or tanh. The hyper



**Fig. 3** The AlexNet architecture without GPU partitioning [12]

parameters(batch size, SGD momentum and learning rate) of AlexNet were set to 128, 0.9 and 0.01.

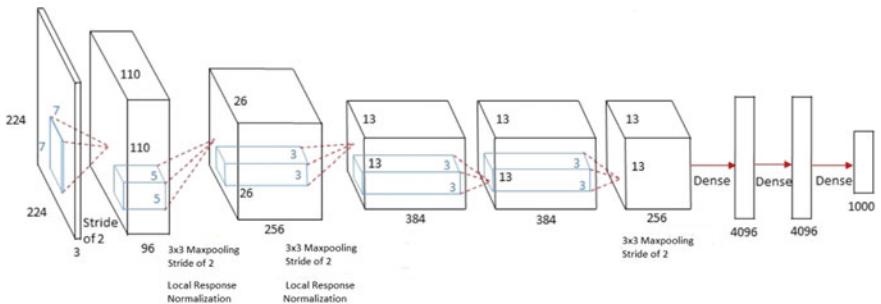
### ZFNet

**ZFNet** architecture the winner of ISLVRC 2013 is almost similar to AlexNet shown in Fig. 4.

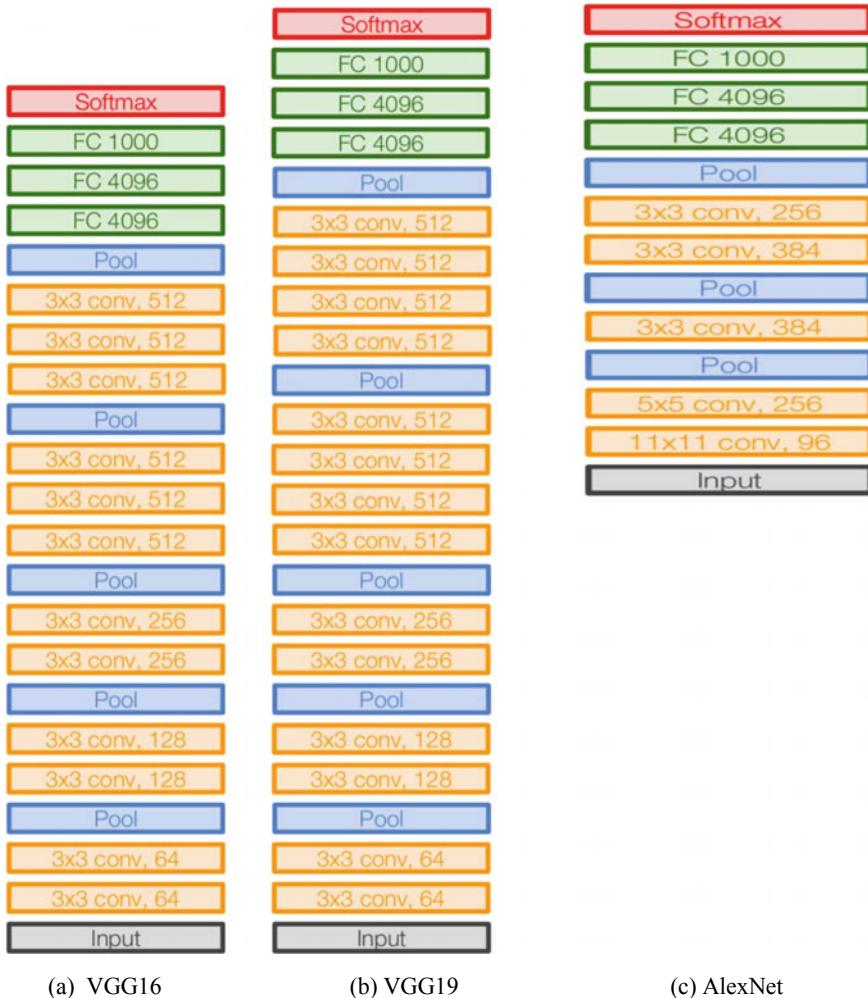
The key difference exists in a few hyper-parameters set and further the architectural changes made were in the first layers filter size and the stride. The filter size of  $11 \times 11$  was reduced to  $7 \times 7$  and the stride of convolution 2 was used instead of stride 4. In convolutional layer 3, 4 and 5 the number of filters used were changed from 384, 384, 256 to 512, 1024, 512 respectively.

### VGGNet

**VGGNet** [15] presented at ILSVRC 2014 is the runner up in the competition which composed of 16 convolutional layers. The architecture seems to be interesting due to its uniform architectural style followed. The same is shown in Fig. 5. The architecture reduced the complexity of using huge filter sizes with huge strides as used in AlexNet, rather throughout the entire network, VGGNet uses a small  $3 \times 3$  filter sizes with stride 1. The model is trained for two to three weeks using 4 GPUs. This architecture is considered as the best model for extracting features from images. It consistently



**Fig. 4** The ZFNet architecture [14]



**Fig. 5** VGGNet architecture, **a, b** 16–19 layers VGGNet, **c** 8 layers AlexNet [15]

uses  $3 \times 3$  as filter size and  $2 \times 2$  as pooling size. The convolution was performed with stride 1 and padding 1 and the pooling was with stride 2. It was observed that the spatial outline of the output volume is always preserved when  $3 \times 3$  filter is applied with a padding of 1, whereas the pooling process compresses the spatial footprint at all times. Hence, the pooling is performed on the non-overlapping spatial regions, and this always reduced the spatial footprint by a factor of 2. This architecture is widely used as a source feature extractor in various applications. The hyper parameter setting of this architecture is publicly available but still it is considered as a challenging architecture due to its usage of 138 million parameters.

### GoogLeNet

The winner of the ILSVRC 2014 competition is the GoogLeNet architecture [16] which achieved a topmost-5 error rate of 6.67%. The architecture is inspired by the LeNet architecture. The novel element included in this GoogLeNet is the inception module. The concepts like image distortions, batch normalization, and RMSprop are used in this architecture [17]. The inception module is developed over a number of very small convolutions to extremely reduce the number of parameters used. A total of 22 deep layers were used in CNN and the parameters are reduced to 4 million from 60 million. The inception model is considered as the central part of the architecture. Figure 6. depicts an example of inception module and also depicts the design of good local network topology where the inception modules are stacked on top of each other.

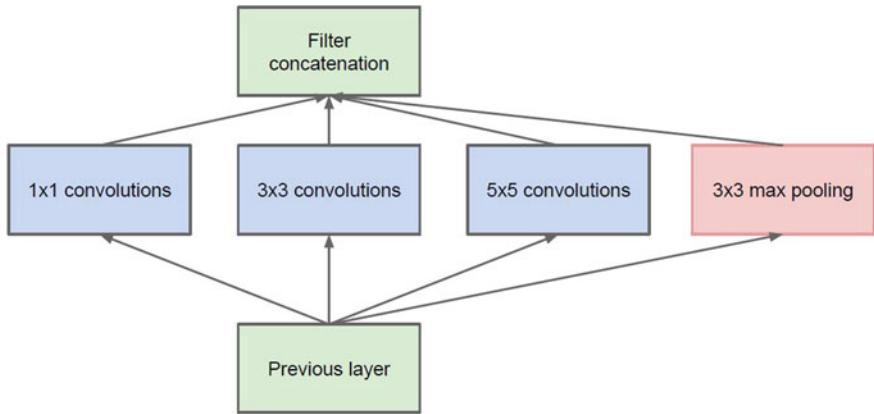
### ResNet

**ResNet** the winner of ILSVRC 2015 [18] trained the network with 152 layers and proved to be having less complexity than VGGNet. The architecture is unique on its own by means of utilizing “skip connections” and also features substantial batch normalization. It achieved a top-5 error rate of 3.57% which was considered as a superior performance than human level predictions on ImageNet data set [19]. The basic unit of this architecture is the residual model which plays a major role in developing whole network by assembling many such residual models (Fig. 7).

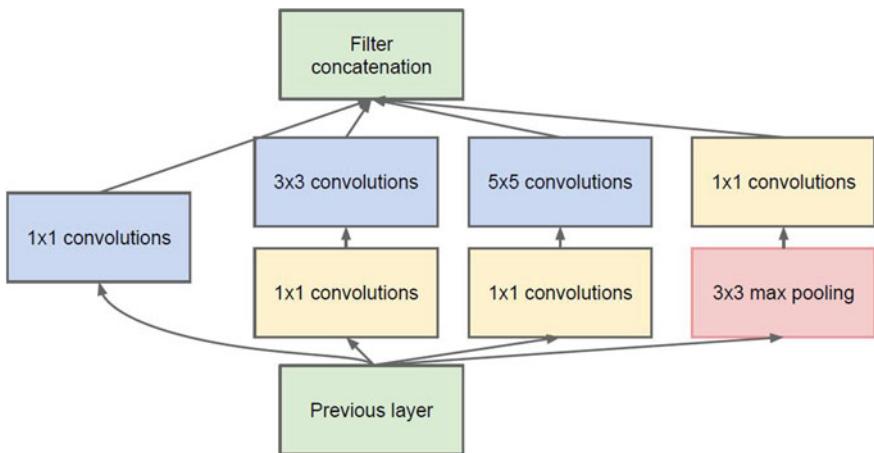
### Recurrent Neural Network

Recurrent Neural Network (RNN) is commonly used along with CNN to employ the concept of recurrence, which is basically using the information from a previous forward pass of the neural network. Figure 8 depicts a simple RNN having a single, self-connected hidden layer. RNNs are more applicable for the applications having input as a sequence. Corresponding to the input sequence, RNN produce either a sequence of outputs or just one output for the entire input sequence. The key concept of RNN [20] is held by the recurrent connections which allow the memory of the previous inputs to carry further in the networks internal state and thus influencing the networks output. There are several variants in using the recurrence relationships. In the first variant, the hidden state for an entity is computed using its corresponding input entity and the previous hidden state. The output of the network is computed using the previous hidden states. The activations functions like tanh are used for the computation of hidden state and softmax functions are used to compute the output of the network. In the second variant of RNN, the hidden state for an entity in the sequence is computed using its corresponding input entity and previous output whereas in the first case it was using the previous hidden state. In the case of RNN producing single output, the computation of hidden state is done for each entity in the input sequence and the output is computed using the last hidden state.

In another variant named Bidirectional RNN [22] in the computation of hidden state, the previous entities information along with the entities that lie further in the sequence are also considered which is not the procedure followed in unidirectional RNNs. Hence, Bidirectional RNNs [Schuster] have both forward hidden state and back ward hidden states. The training of RNN is generally performed by applying a simple unroll operation on the RNN for a given size of input and then training the

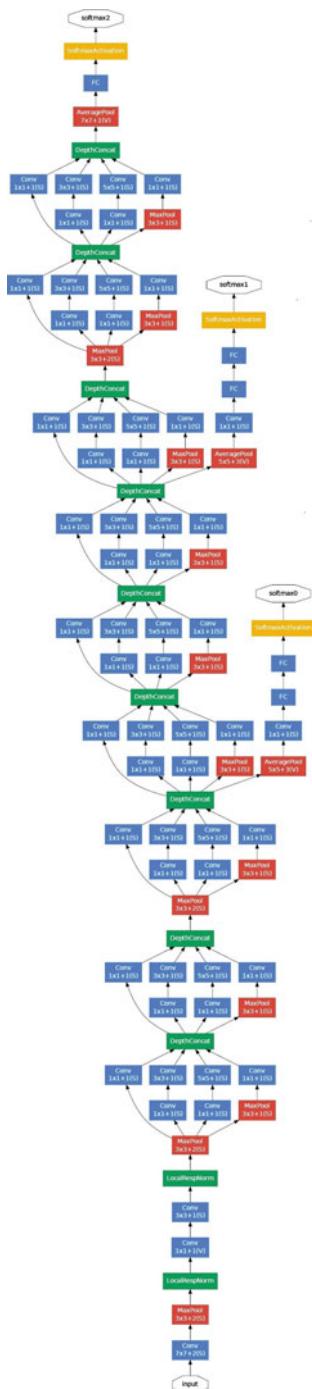


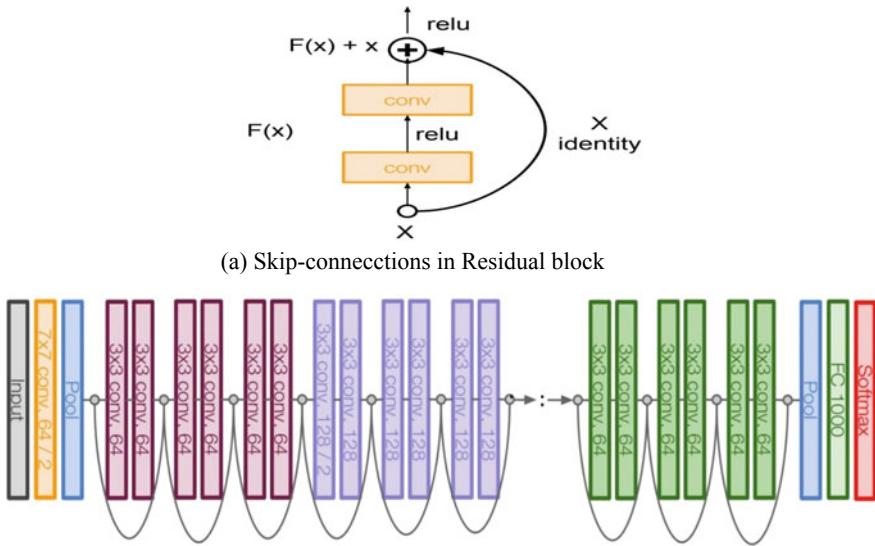
(a) Illustration of Inception module



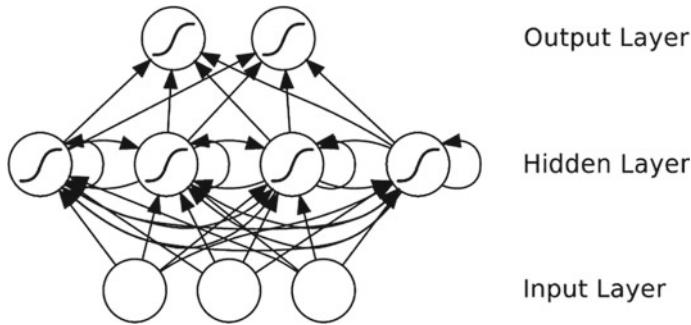
(b) Inception module with dimension reduction

**Fig. 6** Full GoogLeNet architecture (Stack inception modules with dimension reduction placed on top of each other to form GoogLeNet) [16]

**Fig. 6** (continued)



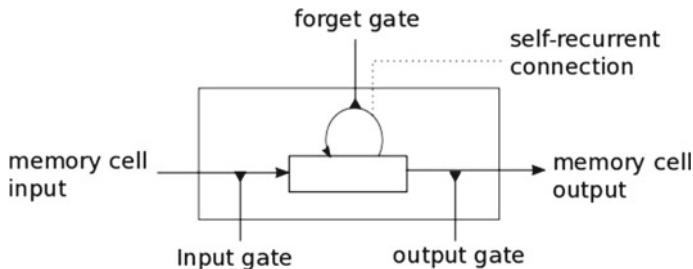
**Fig. 7** Full ResNet architecture (Stack residual blocks) [18]



**Fig. 8** A Recurrent neural network [21]

RNN by computing the gradients and using stochastic gradient descent like technique. When the network is unrolled, each of the input state, hidden state (previous and next) and the output state correspond to a shallow transformation where the transformation is represented as a single layer with a deep multilayer perceptron network.

To overcome the problem of vanishing gradients, a variant of RNN termed as Long Short-Term Memory (LSTM) is proposed. This architecture excels in learning long range sequences and avoids the long term dependency problem [23, 24]. The primary inspiring part of LSTM model is the use of a novel structure called memory cell which consists of four key components namely, an input gate, a neuron with a self-recurrent connection, a forget gate and an output gate. The input gate permits



**Fig. 9** Representation of LSTM memory cell [23]

the incoming signal to change the state of the memory cell or to block it. The self-recurrent connections are assigned with a weight of 1.0 and it guarantees that the position of a memory cell remains stable from one time step to another. The gates in this model are used to control the interactions between the memory cell itself and the environment. The forget gate regulates the self-recurrent connections of the memory cell by allowing the cells to recollect or forget its previous state. Finally, the output gate allows the state of the memory cell to create an impact on other neurons or interrupt it (Fig. 9).

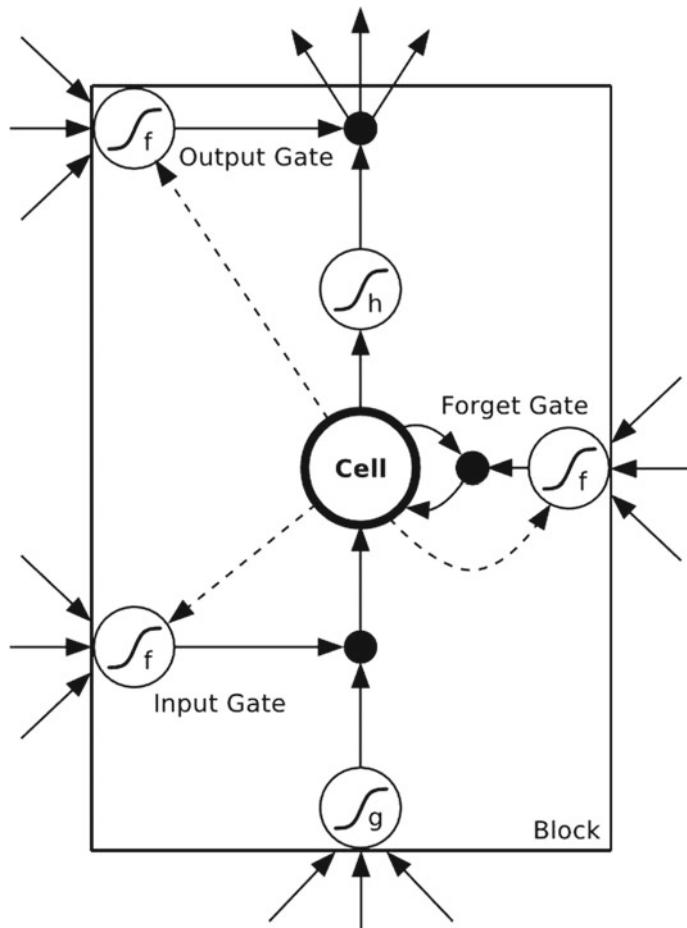
Figure 10 depicts the LSTM memory block with a single cell. Most commonly used gate activation function ‘f’ is the logistic sigmoid and hence the activations are bound to lie between 0 and 1. 0 denotes the gate is closed and 1 denotes the gate is open. ‘tanh’ or logistic sigmoid functions are generally used for cell input and output activation functions. However in some cases identity function is also used as activation function. The dashed lines in the figure denote the weighted peephole connections and the remaining connections in the block are unweighted meaning they have fixed weight of 1.0 [23]. The LSTM network is similar to standard RNN, however the summation units in the hidden layer are replaced by the memory blocks.

### Siamese Neural Networks

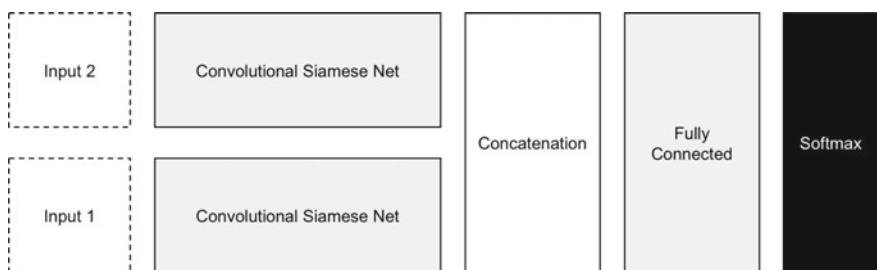
Siamese Neural Network [25] comprises of two or more alike or identical sub networks. The work identical sub network means that they share the same architecture, same parameters and weights. Figure 11 shows the siamese network having the same weights between the networks. Based on the number of sub networks used, the architecture can be termed as pairwise or triplets and accordingly corresponding loss functions are employed. This network is appropriate for person re-identification problem as the output of the network is a similarity score at the top of the network. The network also addresses the data scarcity problem and achieves good recognition rate.

### Activation Functions for Deep Learning Models

The activation function are the crucial part of training deep neural networks. Activation function makes the network more powerful so as to learn complex data and represent the non-linear functional mappings between inputs and outputs. Another



**Fig. 10** LSTM memory block with single cell [23]



**Fig. 11** Siamese Network [25]

important feature of an activation function is that it should be differentiable in order to perform back propagation optimization strategy.

### Sigmoid

Sigmoid activation function is a ‘S’ shaped curve that ranges between 0 and 1. It is defined as shown in Eq. 1.

$$Y = \frac{1}{1 + e^{-x}} \quad (1)$$

Vanishing gradient is a popular issue faced by sigmoid activation functions and this issue is more severe in deep architectures. Moreover, sigmoid activation function is not zero centered. Despite these issues, sigmoid functions are most widely used in many classification tasks.

### Tanh

Hyperbolic Tangent (Tanh) activation function resolves the issue of zero centered in sigmoid function. It ranges between  $-1$  to  $+1$ . The activation function is defined in Eq. 2.

$$Y = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2)$$

Optimization is achieved easily in tanh activation function since it is zero centered. Vanishing gradient problem of sigmoid function still exists in tanh function. Tanh is mostly used in LSTM

### ReLU

Rectified Linear Units (ReLU) emerged as popular activation function in recent years. It is proven to achieve six times improvement in converging when compared to Tanh function. ReLU is defined in Eq. 3.

$$Y = \max(0, x) \quad (3)$$

ReLU [13] is very simple, efficient, and avoids vanishing gradient problem, it is widely used in very deep architectures. However, ReLU2 activation function suffers due to dying ReLU problem where the excessive gradient flowing over a ReLU neuron might affect the weight update in such a way that the neuron never gets activated on any data point. It is limited to use only in hidden layers of deep architectures.

### Leaky ReLU

Leaky ReLU is a kind of solution to overcome the problem of “dying ReLU problem”. The function is designed in such a way that rather than the function being assigned with zero when  $x < 0$ , a leaky ReLU will assign a slight negative slope. The same is defined in Eq. 4.

$$Y = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (4)$$

$\alpha$  value in Leaky ReLU is 0.01. Though leaky ReLU provides good results in few cases it doesn't exhibit consistency at all times.

### Parametric ReLU

Parametric ReLU (PReLU) adaptively learns the parameters of the rectifiers [18], and improves accuracy at negligible extra computational cost. The difference between parametric and leaky ReLU is that leaky ReLU uses a predetermined whereas the parametric ReLU adaptively learns the parameter value from the neural network itself. PReLU is defined as given in Eq. 5.

$$Y = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (5)$$

### Maxout

ReLU and its leaky version are together generalized in Maxout neuron [26] activation function. It has twice the number of parameters. The activation function is defined in Eq. 6.

$$Y = \max(W_1^T x + b_1, W_2^T x + b_2) \quad (6)$$

where  $W_1, W_2$  are weight parameters and  $b_1, b_2$  are bias.

### ELU

Exponential Linear Unit (ELU) [27] is closely related to leaky ReLU. The function has a small slope for negative values and it uses a log curve instead of a straight line. The plus point of ReLU and leaky ReLU are incorporated in ELU. However, for huge negative values, it gets saturated and basically remains inactive. The function is defined in Eq. 7.

$$Y = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases} \quad (7)$$

### Loss Functions Used in Deep Learning Models

The loss functions are primarily to calculate the error of the model. Different loss functions deliver different errors for the same prediction and it has a remarkable effect on the performance of the model. The various loss functions commonly used in person Re-identification problem are placed under two categories, pairwise loss functions and triplet loss functions.

### Pairwise Loss Functions

To describe the pairwise models, consider  $X = \{x_1, x_2, \dots, x_n\}$  and  $Y = \{y_1, y_2, \dots, y_n\}$  as a set of person images and equivalent label for each person. To distinguish the matching from mismatched pairs  $y_i$  and  $y_j$  are compared and accordingly the input images are labeled as matched or mismatched [28] as defined in Eq. 8.

$$image_s(x_i, x_j) = \begin{cases} \text{matching} & \text{if } y_i = y_j \\ \text{mismatching} & \text{if } y_i \neq y_j \end{cases} \quad (8)$$

**Hinge loss** function mostly determines the maximum-margin classification. The function gives an output zero when the distance similarity of the matching pairs is greater than the distance of the mismatching ones with respect to the margin ‘ $m$ ’. The hinge loss function is as defined in Eq. 9.

$$Y = \frac{1}{1 + e^{-x}} \quad (9)$$

**Cosine Similarity loss** function improves or maximizes the cosine value for matching pairs and minimizes the cosine value for the negative pairs when the value is less than margin. The loss function is defined in Eq. (10)

$$image(x_1, x_2, y) = \begin{cases} \max(0, \cos(x_1, x_2) - m) & \text{if } y = 1 \\ 1 - \cos(x_1, x_2) & \text{if } y = -1 \end{cases} \quad (10)$$

**Contrastive loss** function [29] minimizes the mapping function to low dimensional space maps by mapping the similarity of input vectors as output and dissimilarity as distant points. The loss is computed as given in Eq. (11)

$$image(x_1, x_2, y) = (1 - y) \frac{1}{2} (Dist)^2 + (y) \frac{1}{2} \{\max(0, m - Dist)\}^2 \quad (11)$$

In Eq. (11)  $m$  is a margin parameter which is greater than zero and acting as a boundary. The distance between two feature vectors is computed as  $D(x_1, x_2) = \|x_1 - x_2\|_2$ . The average of total loss for each of the pairwise loss functions given above is computed as per Eq. (12)

$$loss(X_1, X_2, Y) = -\frac{1}{n} \sum_{i=1}^n image(x_i^1, x_i^2, y_i) \quad (12)$$

### Triplet Loss Functions

The triplet model consider as set of triplet images. Let  $image_i, image_i^+, image_i^-$  be a set of triplet images where  $image_i$  and  $image_i^+$  are the images of the same person, and  $image_i$  and  $image_i^-$  refer to the images of different persons. The loss function for such models basically creates a margin between the distance metric of matching and mismatching pair and it achieves less distance between the matched pairs and mismatched pairs. Few triplet loss functions are given in this section.

**Euclidean distance** is a commonly used distance metric in pattern recognition models. L2 distance metric is employed in some triplet loss functions denoted as  $dist(W, O_i)$  where  $W = W_i$  is the neural network parameter, and  $F_w(image_i)$  denotes the network output of image  $i$ . Equation (13) measures the distance between similar and dissimilar pair of a single triplet unit  $O_i$

$$dist(W, image_i) = \|F_w(image_i) - F_w(image_i^+)\|^2 - \|F_w(image_i) - F_w(image_i^-)\|^2 \quad (13)$$

**Hinge loss** function aims to reduce the squared hinge loss of the linear SVM, which is same as determining the maximum margin based on the true person match and false person match over training step. This hinge loss function performs a convex approximation in the range of 0–1 ranking error loss which basically approximates the models violation of ranking order specified in the triplet. The loss equation given in (14) has the margin parameter  $g$ . It is a regularization parameter which regularizes the margin between the distance of two image pairs  $(image_i, image_i^+)$  and  $(image_i, image_i^-)$ .  $Dist$  is based on Euclidean distance.

$$loss(image_i, image_i^+, image_i^-) = \max(0, g + Dist(image_i, image_i^+) - Dist(image_i, image_i^-)) \quad (14)$$

Equation (15) is an improved triplet loss function where  $N$  denotes number of triplet training examples,  $\beta$  is a weight factor to balance the inter-class and intra-class constraints. The function  $d(.,.)$  defines the L2-norm distance

$$\begin{aligned} loss(image_i, image_i^+, image_i^-, w) = & \frac{1}{N} \sum (\max\{dist^n(image_i, image_i^+, image_i^-, w), \delta_1\}, \\ & + \beta \max\{dist^p(image_i, image_i^+, image_i^-), \delta_2\}) \end{aligned} \quad (15)$$

**Cross entropy loss or Softmax loss:** This loss function is proposed by McLaughlin et al. [30] and the loss equation is as defined in Eq. (16).

$$image(v) = P(y = c|v) = \frac{\exp(W_c v)}{\sum_n \exp(W_n v)} \quad (16)$$

In Eq. (16),  $v$  is the sequence feature vector,  $n$  is the number of identities,  $y$  is the identity of the person,  $W_c$  and  $W_k$  denote the  $c$ th and  $k$ th column of the softmax weight matrix  $W$ .

**Siamese cost** proposed by Chung et al. [31] for both SpatialNet and TemporalNet architecture is defined in Eq. (17).

$$Dist(\bar{f}_{i_c}, \bar{f}_{j_c}) = \begin{cases} \frac{1}{2} \|\bar{f}_{i_c} - \bar{f}_{j_c}\|^2, & \text{if } i = j \\ \frac{1}{2} \{\max(m - \|\bar{f}_{i_c} - \bar{f}_{j_c}\|, 0)^2, & \text{if } i \neq j \end{cases} \quad (17)$$

In Eq. 16,  $m$  denotes the Siamese margin and  $\bar{f}_{i_c}, \bar{f}_{j_c}$  are the temporally pooled feature vectors for person  $i$  and  $j$ , respectively.

**Binomial deviance loss function** Wu et al. [32] in 2018, proposed to use cosine similarity and the binomial deviance loss function for training the neural network model. The loss function used is given in Eq. (18).

$$\text{loss} = \sum_{i,j} W \odot \ln(\exp^{-\alpha(S-\beta)\odot M} + 1) \quad (18)$$

In Eq. (18),  $\odot$  is the element wise multiplication operator,  $i$  and  $j$  denotes the count of training samples and  $S = [S_{i,j}]_{n \times n}$  represents the similarity matrix for the image pairs having  $n$  representing the total number of training images.  $S_{i,j} = \cos \text{ine}(x_i, x_j)$ .  $\alpha, \beta$  are the hyper parameters. The matrix  $M$  is to encode the training supervision and is defined as

$$M = \begin{cases} 1, & \text{matching pair} \\ -1, & \text{mismatching pair} \end{cases}$$

$W$  represents a weight matrix and is defined as

$$W_{i,j} = \begin{cases} \frac{1}{n_1}; & \text{matching pair} \\ \frac{1}{n_2}; & \text{mismatching pair} \end{cases}$$

$n_1$  and  $n_2$  are the number of matching and mismatching pairs.

### 3 Person Re-identification Datasets

The person re-identification datasets based on image and video that are used in literature are given in Table 1.

### 4 Deep Learning Architectures for Person Re-identification

Different deep learning models used for person Re-Identification are given in Table 2. The details provided are in terms of the architectural style used, activation functions, loss functions and the corresponding re-identification datasets on which the architecture was employed.

#### Evaluation Metrics

The evaluation of the person re-identification models is generally performed using the Cumulative Matching Characteristic (CMC) curve, Synthetic reacquisition rate, and normalized Area under Curve (nAUC). CMC curves are used to evaluate the person re-identification task as a ranking problem [102]. The curve generated is based on the probability of identifying the correct match over the first  $k$  ranks. This

**Table 1** Person Re-identification Datasets

Dataset	Year	# People	# Cameras	Crop image size	# Images	Image/Video	Produced by/Detector
VIPeR [33]	2007	632	2	128 × 128	1264	Image	Hand
ETHZ [34]	2007	146		Vary	4857	Video	Hand
GRID [35]	2009	1025	8	Vary	1275	Image	Hand
QMUL iLIDS [36]	2009	119	2	Vary	476	Image	Hand
3DPeS [37]	2011	200	6	Vary	200,000	Video	Hand
CAVIAR4REID [38]	2011	72	2	17 × 39, 72 × 144	1220	Video	Hand
PRID [39]	2011	385	2	128 × 64		Image	Hand
SAIVT-Softbio [40]	2012	152	8	Vary	64472	Video	Hand
CUHK01 [41]	2012	971	2	160 × 60	3884	Image	Hand
WARD [42]	2012	70	3	128 × 48	4786	Image	Hand
CUHK02 [43]	2013	1816	10	160 × 60	7264	Image	Hand
i-LIDS MCTS [44]	2014		Multiple	Vary	479	Video	Hand
CUHK03 [8]	2014	1360	6	Vary	13,164	Image	DPM [46]/Hand
iLIDS-VID [45]	2014	300	2	Vary	42495	Video	Hand
RAiD [47]	2014	43	4	128 × 64	6920	Image	Hand
Market-1501 [48]	2015	1,501	6	128 × 64	32217	Video	DPM/Hand
PKU-Reid [49]	2016	114	2	128 × 64	1,824	Image	Hand
MARS [50]	2016	1261	6	256 × 128	1191003	Video	DPM + GMMCP
PRW [51]	2016	932	6	Vary	34,304	Video	Hand
CUHK-SYSU [52]	2016	8432		Vary	18,184	Image	Hand
DukeMTMC [53]	2017	1,812	8	Vary	36,411	Video	Doplia

(continued)

**Table 1** (continued)

Dataset	Year	# People	# Cameras	Crop image size	# Images	Image/Video	Produced by/Detector
Airport [54]	2017	9651	6	128 × 64	39902	Video	ACF
MSMT [55]	2018	4,101	15	Vary	126441	Video	Faster RCNN
RPIfield [56]	2018	112	12	Vary	601581	Video	ACF

**Table 2** Deep learning architectures for Person Re-identification using pairwise models

References	Architecture	Activation function	Loss function	Datasets
Zhang et al. [57]	8 layered Deep Convolution Neural Network	Linear SVM (L2-SVM)	Margin based- square hinge loss	VIPeR, Caviar
Yi [7]	5 layered Siamese deep neural network	ReLU	Fisher criterion and binomial deviance cost function	VIPeR, PRID
Li et al. [8]	6 layered Filter pairing neural network (FPNN)	Softmax	Negative log-likelihood cost function	CUHK03, CUHK01, VIPeR, CUHK02
Ahmed et al. [58]	8 layered Deep Convolution Neural Network	Softmax	A stochastic approximation for average loss	CUHK03, CUHK01, VIPeR, CUHK02
Ding et al. [59]	5 layered Deep Convolution Neural Network	ReLU	The triplet-based loss function	iLIDS, VIPeR
Zhang et al. [60]	Bit-Scalable Deep Hashing Framework (10 layers)	tanh	The triplet-based loss function	MNIST, CIFAR-10, CIFAR-20, NUS-WIDE.
Shi et al. [61]	Convolutional neural networks with Mahalanobis metric layers	ReLU	Mahalanobis distance	CUHK03, CUHK01, VIPeR
Iodice et al. [62]	Strict Pyramidal Deep CNN architecture	tanh	Cross entropy loss function	VIPeR
Cheng et al. [63]	Multi-channel Convolution Neural Network (11 layers)	ReLU	The triplet-based loss function	i-LIDS, VIPeR, PRID2011, CUHK01
Chen et al. [64]	Deep convolutional neural network with Deep ranking framework	Softmax	Logistic loss function	VIPeR, CUHK-01, CAVIAR4REID
Wu et al. [65]	Fusion Feature Net (CNN and Ensemble of Local Features)	The ReLU and dropout	Softmax loss and Cross-entropy loss	VIPeR, CUHK01, PRID450s

(continued)

**Table 2** (continued)

References	Architecture	Activation function	Loss function	Datasets
Xiao et al. [66]	Convolutional Neural Network with Domain Guided Dropout layer	Softmax	Softmax loss and Cross-entropy loss	CUHK01, CUHK03, PRID, VIPeR
Wu et al. [67]	10 layered Deep Convolutional Neural Networks	tanh	RMSprop	CUHK03, Market-1501, CUHK01.
Li et al. [68]	Siamese Convolutional Neural Network with Null Foley-Sammon Transform (NFST) metric learning	ReLU	Contrastive loss function	Market-1501
Shi et al. [69]	7 layered Deep Convolutional Neural Networks (NFST) metric learning	ReLU and Linear activation	Moderate Positive Mining Method	CUHK03, CUHK01, VIPeR
Varior et al. [70]	Siamese Long Short-Term Memory (LSTM) Architecture	Sigmoid and tanh	Contrastive loss function	Market-1501, CUHK03, VIPeR
Wang et al. [71]	Deep CNN for Single-image and Cross-image Representations	ReLU	Pairwise and triplet comparison models	CUHK03, CUHK01, VIPeR
Franco and Oliveira [72]	Hybrid network composed of Convolutional Neural Network and Deep Belief Network Denoising Autoencoder.	Softmax	Euclidean distance	i-LIDS, CUHK01, CUHK03, VIPeR
McLaughlin et al. [30]	Siamese network architecture using RNN and CNN for video-based person re-identification.	tanh	Cross-entropy loss, or softmax function,	iLIDS-VID, PRID-2011

(continued)

**Table 2** (continued)

References	Architecture	Activation function	Loss function	Datasets
Liu et al. [74]	A Multi-scale triplet convolutional neural Network		Comparative similarity loss	Market1501, CUHK Campus, VIPeR, RID2011
Wang et al. [75]	Multiscale and Multipart deep neural Network	ReLU	Adaptive margin listwise loss	CUHK03, CUHK01, VIPeR
Liu et al. [76]	Comparative Attention Network (CAN) based architecture	Softmax	Triplet loss and identification loss (the corresponding softmax loss function)	VIPeR, CUHK03, Market-1501, CUHK01
Wu et al. [77]	Hybrid deep architectures composed of Fisher vectors and deep neural networks	ReLU	LDA based loss function.	VIPeR, CUHK03, CUHK01, Market 1501
McLaughlin et al. [73]	Siamese deep neural network with multitask learning	tanh	Euclidean distance	VIPeR, i-LIDS, 3DPeS, CAVIAR, PETA
Su et al. [78]	Pose-driven Deep Convolutional Network	tanh	Softmax Loss	CUHK03, Market 1501, VIPeR
Franco and Oliveira [79]	A hybrid network having three CNNs and a DBN-DAE is proposed for learning convolutional covariance features	Softmax	Cross entropy error/Euclidean distance	VIPeR, i-LIDS, CUHK03
Qian et al. [80]	Multi-scale deep learning model based on Siamese network	Softmax	Verification loss and classification loss	CUHK03, CUHK01, VIPeR
Zhu et al. [81]	A light convolution neural with hybrid similarity function for metric learning.	ReLU	Log-logistic model	QMUL GRID, VIPeR, CUHK03

(continued)

**Table 2** (continued)

References	Architecture	Activation function	Loss function	Datasets
Cheng et al. [82]	Embedding the structured graph Laplacian with softmax loss in Deep CNN model.	ReLU	Joint loss function: softmax loss together with structured graph Laplacian Embedding	3DPES, CUHK01, CUHK03, Market-1501.
Mao et al. [83]	Deep convolutional neural networks with Pyramid Matching Module.	Softmax	Cross-entropy loss	CUHK03, CUHK01, VIPeR
Li et al. [84]	Multi-Scale Context-Aware Network (MSCAN) and Spatial Transform Networks (STN) with CNN	ReLU	Softmax Loss	Market1501, CUHK03, MARS
Lin et al. [85]	A consistent-aware CNN based deep learning architecture	ReLU	Contrastive loss function	Market-1501, RAID, WARD
Chung et al. [31]	Two Stream Siamese Convolutional Neural Network	tanh	Siamese cost and Identification cost function.	PRID2011, iLIDS-VID
Bai et al. [86]	Long Short-Term Memory (LSTM) based end-to-end deep architecture	Softmax	Softmax and Triplet Loss	Market-1501, CUHK03, DukeMTMC
Chang et al. [87]	ResNet architecture to extract pool5 CNN features.	ReLU	Triplet loss	i-LIDS, PRID-2011, CUHK03, Market-1501.
Chen et al. [88]	Deep CNN model with Local Maximal Occurrence features.	ReLU	Triplet loss and weighted cross entropy loss function.	VIPeR, PETA, APIS, CUHK03.
Tao et al. [89]	Multi-view feature learning using Deep convolution neural network.	ReLU	The triplet-based loss function.	VIPeR, GRID.

(continued)

**Table 2** (continued)

References	Architecture	Activation function	Loss function	Datasets
Wu et al. [90]	Deep hashing framework with Convolutional Neural Networks	Sigmoid	Structured loss function	CUHK03, Market-1501.
Su et al. [91]	Multi-type attribute learning based on deep CNN architecture.	Softmax	Attributes triplet loss	PETA, VIPeR, GRID, Market, CUHK03.
Wang et al. [92]	Deep CNN with large adaptive margin loss function.	ReLU	Adaptive margin loss function	PRID2011, Market1501, CUHK01, 3DPeS
Wu et al. [93]	Combination of two CNN and four directional RNN model with multiplicative integration gating function	ReLU	Cosine similarity function and binomial deviance loss function	VIPeR, CUHK03, Market 1501
Zhang et al. [94]	Deep integrated CNN-based architecture.	Softmax	Soft triplet loss and cross entropy loss	Market 1501, CUHK03, DukeMTMC.
Liu et al. [95]	Deep CNN model with Gaussian of Gaussian (GOG) features.	Softmax	Identification loss and verification loss function	Market1501, VIPeR, VeRi.
Wang et al. [96]	Feature fusion & metric learning based on Deep CNN with Null Foley-Sammon transform (NFST)	–	Null Foley-Sammon transform	VIPeR, CUHK01,
Yuan et al. [97]	Deep convolutional neural network (DCNN) regularized by both the improved triplet loss and softmax loss.	ReLU	Triplet loss function and Softmax loss function.	Market-1501, CUHK03, CUHK01, PRID, 3DPeS, VIPeR, iLIDS.

(continued)

**Table 2** (continued)

References	Architecture	Activation function	Loss function	Datasets
Xin et al. [98]	Heterogeneous convolutional neural networks with multi-view clustering method	–	Identification loss and verification loss functions.	Market-1501, DukeMCMCT, CUHK03.
Wu et al. [99]	Triplet-based CNN model with Online Instance Matching (OIM) loss.	–	Triplet loss and Online Instance Matching (OIM) loss	DukeMTMC-reID, CUHK03, CUHK01 and Market-1501
Zhou et al. [100]	Multi-camera Generative adversarial networks (GAN)	–	Adversarial loss, Domain classification loss, Reconstruction loss, ID consistent Loss.	Market-1501, CUHK03, DukeMTMC-ReID
Zhong et al. [101]	Recurrent Comparative Network with multilevel CNN features.	tanh	Cross-entropy loss, recognition loss, soft-max classification loss.	CUHK03, CUHK01, Market1501, DukeMTMC.

measure can also be termed as recall at  $k$ . The curve plot is the probability of correct match that is ranked equal to or less than a particular threshold against the size of the gallery set. The two aspects of the CMC curve are the first rank re-identification rate and the steep of the curve. The steeper the curve is the better the performance is. Secondly, Synthetic Reacquisition Rate (SRR) curve is derived from the CMC curve and it measures the probability that any of the  $k$  best matches is correct. The Normalized area under the CMC curve (nAUC) provides the overall performance by having the model yielding a positive match over a negative match. Higher the value of nAUC is, the better the performance would be. The main objective of all the re-identification models is to improve on Rank-1 recognition rates.

## 5 Experimental Setup

The Market 1501 dataset consists of 32,668 annotated bounding boxes and 1501 identities captured by 6 cameras, 5 of which are high resolution and 1 is low resolution. Each identity or person appears in at least 2 cameras. It is the largest and most robust open source re identification dataset available online.

The dataset employs the Deformable part model in order to detect pedestrians in the images. For each detected bounding box, a hand drawn ground truth bounding box is created and the intersection over union is calculated. If the IoU value is greater than 50%, the bounding box generated is marked as good, if it is over 20% then it is marked as distractor, else otherwise it is marked as junk.

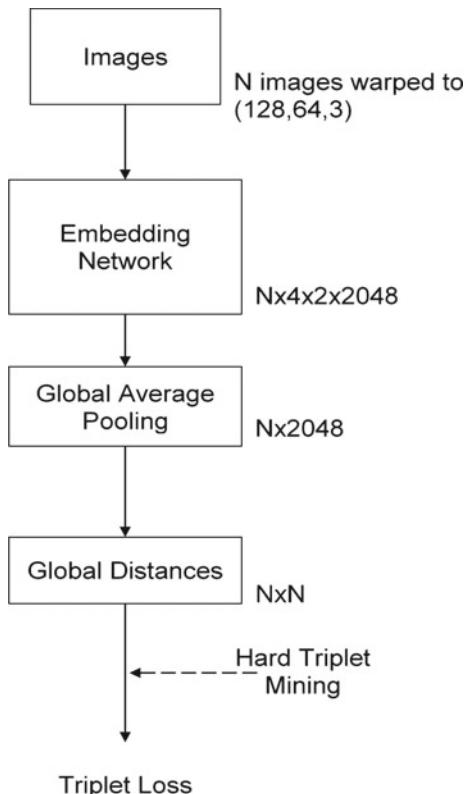
The setup contains a base, pre trained CNN as an embedding network to produce vector embeddings of the Images in n-Dimensional space. In our experiments, Resnet and Xception networks are used as the CNN to extract said feature vectors which are pretrained on the ImageNet dataset (Fig. 12).

The current model focuses on embedding the images into an n-Dimensional vector space, a process essential to achieve re identification. Once this embedding network is trained it can be fed validation images which will be mapped into the vector space such that the vectors representation of the same person starts forming clusters. Then these clusters can be extracted using clustering algorithms like K-Means clustering in order to achieve a complete end to end re-identification system. This experimental setup considers only the first half of the Re-id process involving the embedding of the images into the vector space.

The first experimental set up consists of a residual network with 50 layers, used as the embedding network. The images are fed to the ResNet to obtain embedding's of the images. The obtained results are then passed to a global average pooling layer to reduce it into a one-dimensional vector. Then, online hard mining is carried out to mine the hardest triplet in each batch. These triplet vectors are used to calculate the triplet loss as

$$L = \max[d(a, p) - d(a, n) + margin, 0] \quad (19)$$

**Fig. 12** Architecture used for vector mapping of images

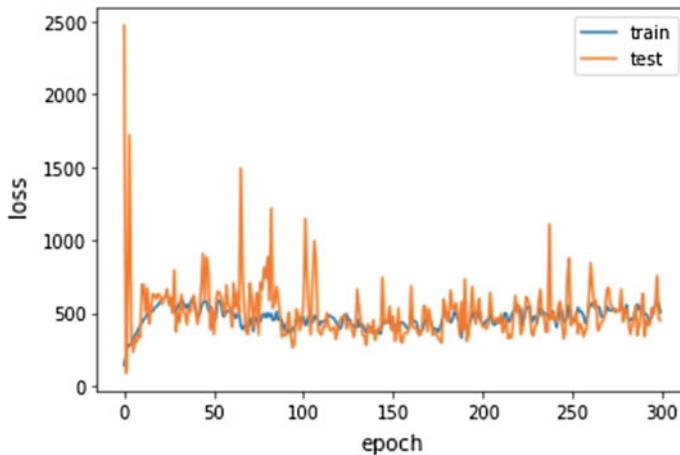


where  $d(x, y)$  is the distance between the embedding's of  $x$  and  $y$ ,  $a$  is the anchor image,  $p$  is the positive image and  $n$  is the negative image and margin is a tunable hyper parameter. The network has a total of 23,587,712 parameters of which 23,534,592 are trainable and 53,120 are non-trainable.

The second experiment used Xception networks with modified depth wise separable convolutions as the embedding network. The architecture has 36 convolutional layers which form the base for feature extraction. The embedding's produced are passed on to a global average pooling layer and the vectors so produced are mined for hard triplets which are then used to calculate triplet loss. The network has a total of 20,861,480 parameters of which 20,806,952 are trainable and 54,528 are non-trainable.

Resnet 50 and Xception networks are currently the highest performing networks on the ImageNet dataset and hence are used as feature extractors to embed the image dataset into an n-Dimensional space. We use pretrained models, with ImageNet weights for the embedding network.

Adadelta with the learning rate set to 1.0 is used as the optimizer with the parameters like rho (decay factor) and the decay set to 0. A network trained using this



**Fig. 13** Loss graph for architecture with Resnet50 for embedding network

method can be used to produce image vectors that can then be passed through a clustering algorithm to achieve re identification.

The experiment was carried out on a Linux system with 16 GB RAM, Core i7-8700 k processor and an NVIDIA Titan Xp graphics card with 12 GBs of VRAM.

The obtained results are summarized in the form of graphs which contain the number of epochs on the x-axis and loss value on training and validation given in y-axis.

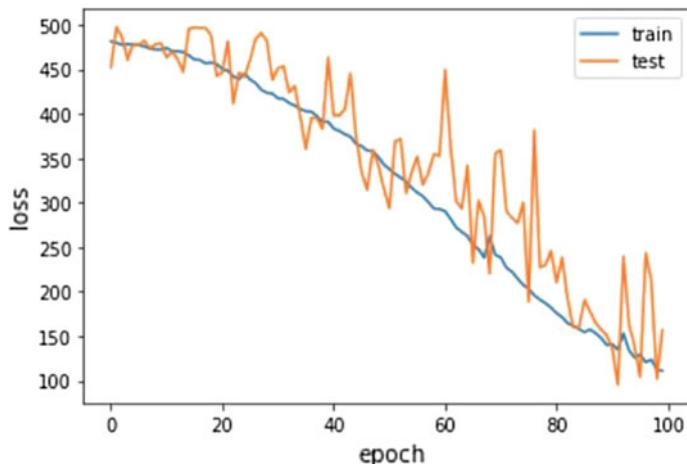
In the first experiment, the features extracted from ResNet50, when used to train the model were unable to converge to a satisfactory degree after running for 300 epochs. The minimum validation loss obtained was in the initial phases of the training and was of the magnitude 147.2. The loss then proceeds to diverge despite using smaller lower learning rates and also while using other optimizers (Fig. 13).

Subsequently in the second experiment, the features extracted using Xception networks, when used in the architecture described before were able to converge in 100 epochs to about 80.6 without overfitting the training data (Fig. 14).

Comparing the two, we see that a network trained with an Xception Network as the embedding network performs better than a network that uses a Resnet50 for the embedding network.

### Advantages of Deep Learning Models Towards Person Re-identification

1. Deep learning models attempts to learn the high level features in incremental manner.
2. Automatic feature learning eliminates the need of domain expert and the need for hard crafted features in person Re-identification
3. During both training and testing time, generally deep learning algorithms works faster.



**Fig. 14** Loss graph for architecture with Xception net for embedding network

### Limitations of Deep Learning Techniques Towards Person Re-identification

1. Having less number of training samples is a major issue in person Re-identification problem. Need large datasets to develop robust models which can handle pose and viewpoint variations in the images.
2. To train the deep learning models with in a reasonable amount of time, a high end infrastructure is required.
3. The processing time is neglected in most of the works done. Hence, we require minimum size architectures with good recognition rates.
4. Need some more research work to be carried out to get 100% recognition rates and to deal with anomalies.

## 6 Conclusions and Future Work

Person Re-identification is considered to be a challenging task in the CCTV surveillance system. Though the machine learning techniques have been recognized as good performers for person re-identification, deep learning technique play a vital role for this problem as it reduces much of human interference and the recognition rate is high in deep convolutional neural networks. Despite its high importance, there exists lot of issues in implementing the system in real world scenario. The challenging task would be the architecture size in terms of number of parameters and layers without affecting the recognition rate. The future work would be to focus more on scalable and end to end re-identification systems which can work in real time scenarios.

**Acknowledgements** The authors thank VIT for providing ‘VIT SEED GRANT’ for carrying out this research work. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research on person Re-identification.

## References

1. Bedagkar-Gala, A., Shah, S.K.: A survey of approaches and trends in person re-identification. *Image Vis. Comput.* **32**(4), 270–286 (2014)
2. Zheng, L., Yang, Y., Hauptmann, A. G.: Person re-identification: Past, present and future (2016). arXiv preprint [arXiv:1610.02984](https://arxiv.org/abs/1610.02984)
3. Saghafi, M.A., Hussain, A., Zaman, H.B., Saad, M.H.M.: Review of person re-entification techniques. *IET Comput. Vision* **8**(6), 455–474 (2014)
4. Zajdel, W., Zivkovic, Z., Krose, B.J.A.: Keeping track of humans: have I seen this person before? In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005, pp. 2081–2086. IEEE (2005)
5. Gheissari, N., Sebastian, T.B., Hartley, R.: Person reidentification using spatiotemporal appearance. In: Null, pp. 1528–1535. IEEE (2006)
6. Bazzani, L., Cristani, M., Perina, A., Farenzena, M., Murino, V.: Multiple-shot person re-identification by hpe signature. In: 20th International Conference on Pattern Recognition (ICPR), 2010, pp. 1413–1416. IEEE (2010)
7. Yi, D., Lei, Z., Liao, S., Li, S.Z.: Deep metric learning for person re-identification. In: Pattern Recognition (ICPR), 2014 22nd International Conference on, pp. 34–39. IEEE (2014)
8. Li, W., Zhao, R., Xiao, T., Wang, X.: Deepreid: deep filter pairing neural network for person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 152–159 (2014)
9. Xu, Y., Ma, B., Huang, R., Lin, L.: Person search in a scene by jointly modeling people commonness and person uniqueness. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 937–940. ACM (2014)
10. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: ISCAS Vol. 2010, pp. 253–256 (2010)
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
12. Charu, C.A.: Neural Networks and Deep Learning: A Textbook. Springer (2019)
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
14. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European Conference on Computer Vision, pp. 818–833. Springer, Cham (2014)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)
16. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D.,..., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
17. Dauphin, Y.N., de Vries, H., Chung, J., Bengio, Y.: RMSProp and equilibrated adaptive learning rates for non-convex optimization. CoRR [arXiv:1502.04390](https://arxiv.org/abs/1502.04390) (2015)
18. He, K., Zhang, X., RenSchustera, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
19. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Berg, A.C.: Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* **115**(3), 211–252 (2015)

20. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
21. Graves, A.: Supervised sequence labelling. In: Supervised Sequence Labelling with Recurrent Neural Networks, pp. 5–13. Springer, Berlin, Heidelberg (2012)
22. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **45**(11), 2673–2681 (1997)
23. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* (1994)
24. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
25. Shanmugamani, R.: Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras. Packt Publishing Ltd (2018)
26. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks (2013). arXiv preprint [arXiv:1302.4389](https://arxiv.org/abs/1302.4389)
27. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus) (2015). arXiv preprint [arXiv:1511.07289](https://arxiv.org/abs/1511.07289)
28. Farenzena, M., Bazzani, L., Perina, A., Murino, V., Cristani, M.: Person re-identification by symmetry-driven accumulation of local features. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010, pp. 2360–2367. IEEE (2010)
29. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: Null, pp. 1735–1742. IEEE (2006)
30. McLaughlin, N., Martinez del Rincon, J., Miller, P.: Recurrent convolutional network for video-based person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1325–1334 (2016)
31. Chung, D., Tahboub, K., Delp, E.J.: A two stream siamese convolutional neural network for person re-identification. In: The IEEE International Conference on Computer Vision (ICCV) (2017)
32. Wu, L., Wang, Y., Li, X., Gao, J.: What-and-where to match: deep spatially multiplicative integration networks for person re-identification. *Pattern Recogn.* **76**, 727–738 (2018)
33. Gray, D., Tao, H.: Viewpoint invariant pedestrian recognition with an ensemble of localized features. In: European Conference on Computer Vision, pp. 262–275. Springer, Berlin, Heidelberg (2008)
34. Ess, A., Leibe, B., Van Gool, L.: Depth and appearance for mobile scene analysis. In: 11th International Conference on Computer Vision, 2007. ICCV 2007. IEEE, pp. 1–8. IEEE (2007)
35. Loy, C.C., Xiang, T., Gong, S.: Time-delayed correlation analysis for multi-camera activity understanding. *Int. J. Comput. Vision* **90**(1), 106–129 (2010)
36. Zheng, W., Gong, S., Xiang., T.: Associating groups of people. In: BMVC (2009)
37. Baltieri, D., Vezzani, R., Cucchiara, R.: 3dps: 3d people dataset for surveillance and forensics. In: Proceedings of the 2011 Joint ACM Workshop on Human Gesture and Behavior Understanding, pp. 59–64. ACM (2011)
38. Cheng, D.S., Cristani, M., Stoppa, M., Bazzani, L., Murino, V.: Custom pictorial structures for re-identification. In: Bmvc, Vol. 1, No. 2, p. 6 (2011)
39. Hirzer, M., Beleznai, C., Roth, P. M., Bischof, H.: Person re-identification by descriptive and discriminative classification. In: Scandinavian Conference on Image Analysis, pp. 91–102. Springer, Berlin, Heidelberg (2011)
40. Bialkowski, A., Denman, S., Sridharan, S., Fookes, C., Lucey, P.: A database for person re-identification in multi-camera surveillance networks. In: 2012 International Conference on Digital Image Computing Techniques and Applications (DICTA), pp. 1–8. IEEE (2012)
41. Li, W., Zhao, R., Wang, X.: Human reidentification with transferred metric learning. In: Asian Conference on Computer Vision, pp. 31–44. Springer, Berlin, Heidelberg (2012)
42. Martinel, N., Micheloni, C.: Re-identify people in wide area camera network. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2012, pp. 31–36. IEEE (2012)
43. Li, W., Wang, X.: Locally aligned feature transforms across views. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3594–3601 (2013)

44. Wang, T., Gong, S., Zhu, X., Wang, S.: Person re-identification by video ranking. In: European Conference on Computer Vision, pp. 688–703. Springer, Cham (2014)
45. Branch, H.O.S.D.: Imagery library for intelligent detection systems (i-lids). In: The Institution of Engineering and Technology Conference on Crime and Security, pp. 445–448 (2006)
46. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.* **32**(9), 1627–1645 (2010)
47. Das, A., Chakraborty, A., Roy-Chowdhury, A.K.: Consistent re-identification in a camera network. In: European Conference on Computer Vision, pp. 330–345. Springer, Cham (2014)
48. Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., Tian, Q.: Scalable person re-identification: a benchmark. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1116–1124 (2015)
49. Ma, L., Liu, H., Hu, L., Wang, C., Sun, Q.: Orientation driven bag of appearances for person re-identification (2016). arXiv preprint [arXiv:1605.02464](https://arxiv.org/abs/1605.02464)
50. Zheng, L., Bie, Z., Sun, Y., Wang, J., Su, C., Wang, S., Tian, Q.: Mars: a video benchmark for large-scale person re-identification. In: European Conference on Computer Vision, pp. 868–884. Springer, Cham (2016)
51. Zheng, L., Zhang, H., Sun, S., Chandraker, M., Yang, Y., Tian, Q.: Person re-identification in the wild. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1367–1376 (2017)
52. Xiao, T., Li, S., Wang, B., Lin, L., Wang, X.: End-to-end deep learning for person search (2016). arXiv preprint [arXiv:1604.01850](https://arxiv.org/abs/1604.01850), **1**(2)
53. Ristani, E., Solera, F., Zou, R., Cucchiara, R., Tomasi, C.: Performance measures and a data set for multi-target, multi-camera tracking. In: European Conference on Computer Vision, pp. 17–35. Springer, Cham (2016)
54. Camps, O., Gou, M., Hebble, T., Karanam, S., Lehmann, O., Li, Y.,..., Xiong, F.: From the lab to the real world: Re-identification in an airport camera network. *IEEE Trans. Circuits Syst. Video Technol.* **27**(3), 540–553 (2017)
55. Wei, L., Zhang, S., Gao, W., Tian, Q.: Person transfer gan to bridge domain gap for person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 79–88 (2018)
56. Zheng, M., Karanam, S., Radke, R.J.: RPIfield: a new dataset for temporally evaluating person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 1893–1895 (2018)
57. Zhang, G., Kato, J., Wang, Y., Mase, K.: People re-identification using deep convolutional neural network. In: Computer Vision Theory and Applications (VISAPP), 2014 International Conference on Vol. 3, pp. 216–223. IEEE (2014)
58. Ahmed, E., Jones, M., Marks, T.K.: An improved deep learning architecture for person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3908–3916 (2015)
59. Ding, S., Lin, L., Wang, G., Chao, H.: Deep feature learning with relative distance comparison for person re-identification. *Pattern Recogn.* **48**(10), 2993–3003 (2015)
60. Zhang, R., Lin, L., Zhang, R., Zuo, W., Zhang, L.: Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Trans. Image Process.* **24**(12), 4766–4779 (2015)
61. Shi, H., Zhu, X., Liao, S., Lei, Z., Yang, Y., Li, S.Z.: Constrained deep metric learning for person re-identification (2015). arXiv preprint [arXiv:1511.07545](https://arxiv.org/abs/1511.07545)
62. Iodice, S., Petrosino, A., Ullah, I.: Strict pyramidal deep architectures for person re-identification. In: International Workshop on Neural Networks, pp. 179–186. Springer, Cham (2015)
63. Cheng, D., Gong, Y., Zhou, S., Wang, J., Zheng, N.: Person re-identification by multi-channel parts-based cnn with improved triplet loss function. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1335–1344 (2016)

64. Chen, S.Z., Guo, C.C., Lai, J.H.: Deep ranking for person re-identification via joint representation learning. *IEEE Trans. Image Process.* **25**(5), 2353–2367 (2016)
65. Wu, S., Chen, Y.C., Li, X., Wu, A.C., You, J.J., Zheng, W.S.: An enhanced deep feature representation for person re-identification. In: Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on, pp. 1–8. IEEE (2016)
66. Xiao, T., Li, H., Ouyang, W., Wang, X.: Learning deep feature representations with domain guided dropout for person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1249–1258 (2016)
67. Wu, L., Shen, C., Hengel, A.V.D.: Personnet: Person re-identification with deep convolutional neural networks (2016). arXiv preprint [arXiv:1601.07255](https://arxiv.org/abs/1601.07255)
68. Li, S., Liu, X., Liu, W., Ma, H., Zhang, H.: A discriminative null space based deep learning approach for person re-identification. In: 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), 2016, pp. 480–484. IEEE (2016)
69. Shi, H., Yang, Y., Zhu, X., Liao, S., Lei, Z., Zheng, W., Li, S.Z.: Embedding deep metric for person re-identification: A study against large variations. In: European Conference on Computer Vision, pp. 732–748. Springer, Cham (2016)
70. Varior, R.R., Shuai, B., Lu, J., Xu, D., Wang, G.: A siamese long short-term memory architecture for human re-identification. In: European Conference on Computer Vision, pp. 135–153. Springer, Cham (2016)
71. Wang, F., Zuo, W., Lin, L., Zhang, D., Zhang, L.: Joint learning of single-image and cross-image representations for person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1288–1296 (2016)
72. Franco, A., Oliveira, L.: A coarse-to-fine deep learning for person re-identification. In: IEEE Winter Conference on Applications of Computer Vision (WACV), 2016, pp. 1–7. IEEE (2016)
73. McLaughlin, N., del Rincon, J.M., Miller, P.C.: Person reidentification using deep convnets with multitask learning. *IEEE Trans. Circuits Syst. Video Techn.* **27**(3), 525–539 (2017)
74. Liu, J., Zha, Z.J., Tian, Q.I., Liu, D., Yao, T., Ling, Q., Mei, T.: Multi-scale triplet cnn for person re-identification. In: Proceedings of the 2016 ACM on Multimedia Conference, pp. 192–196. ACM (2016)
75. Wang, J., Wang, Z., Gao, C., Sang, N., Huang, R.: DeepList: learning deep features with adaptive Listwise constraint for person reidentification. *IEEE Trans. Circuits Syst. Video Techn.* **27**(3), 513–524 (2017)
76. Liu, H., Feng, J., Qi, M., Jiang, J., Yan, S.: End-to-end comparative attention networks for person re-identification. *IEEE Trans. Image Process.* **26**(7), 3492–3506 (2017)
77. Wu, L., Shen, C., van den Hengel, A.: Deep linear discriminant analysis on fisher networks: A hybrid architecture for person re-identification. *Pattern Recogn.* **65**, 238–250 (2017)
78. Su, C., Li, J., Zhang, S., Xing, J., Gao, W., Tian, Q.: Pose-driven deep convolutional model for person re-identification. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 3980–3989. IEEE (2017)
79. Franco, A., Oliveira, L.: Convolutional covariance features: Conception, integration and performance in person re-identification. *Pattern Recogn.* **61**, 593–609 (2017)
80. Qian, X., Fu, Y., Jiang, Y.G., Xiang, T., Xue, X.: Multi-scale deep learning architectures for person re-identification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5399–5408 (2017)
81. Zhu, J., Zeng, H., Liao, S., Lei, Z., Cai, C., Zheng, L.: Deep hybrid similarity learning for person re-identification. *IEEE Trans. Circuits Syst. Video Technol.* **28**(11), 3183–3193 (2018)
82. Cheng, D., Gong, Y., Chang, X., Shi, W., Hauptmann, A., Zheng, N.: Deep Feature Learning via Structured Graph Laplacian Embedding for Person Re-Identification. *Pattern Recogn.* (2018)
83. Mao, C., Li, Y., Zhang, Z., Zhang, Y., Li, X.: Pyramid Person Matching Network for Person Re-identification(2018). arXiv preprint [arXiv:1803.02547](https://arxiv.org/abs/1803.02547)
84. Li, D., Chen, X., Zhang, Z., Huang, K.: Learning deep context-aware features over body and latent parts for person re-identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 384–393 (2017)

85. Lin, J., Ren, L., Lu, J., Feng, J., Zhou, J.: Consistent-aware deep learning for person re-identification in a camera network. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 6 (2017)
86. Bai, X., Yang, M., Huang, T., Dou, Z., Yu, R., Xu, Y.: Deep-Person: Learning Discriminative Deep Features for Person Re-Identification (2017). arXiv preprint [arXiv:1711.10658](https://arxiv.org/abs/1711.10658)
87. Chang, Y. S., Wang, M.Y., He, L., Lu, W., Su, H., Gao, N., Yang, X.A.: Joint deep semantic embedding and metric learning for person re-identification. Pattern Recogn. Lett. (2018)
88. Chen, Y., Duffner, S., Stoian, A., Dufour, J.Y., Baskurt, A.: Deep and low-level feature-based attribute learning for person re-identification. Image Vis. Comput. **79**, 25–34 (2018)
89. Tao, D., Guo, Y., Yu, B., Pang, J., Yu, Z.: Deep multi-view feature learning for person re-identification. IEEE Trans. Circuits Syst. Video Technol. **28**(10), 2657–2666 (2018)
90. Wu, L., Wang, Y., Ge, Z., Hu, Q., Li, X.: Structured deep hashing with convolutional neural networks for fast person re-identification. Comput. Vis. Image Underst. **167**, 63–73 (2018)
91. Su, C., Zhang, S., Xing, J., Gao, W., Tian, Q.: Multi-type attributes driven multi-camera person re-identification. Pattern Recogn. **75**, 77–89 (2018)
92. Wang, J., Zhou, S., Wang, J., Hou, Q.: Deep ranking model by large adaptive margin learning for person re-identification. Pattern Recogn. **74**, 241–252 (2018)
93. Wu, D., Zheng, S.J., Yuan, C.A., Huang, D.S.: A deep model with combined losses for person re-identification. Cogn. Syst. Res. **54**, 74–82 (2019)
94. Zhang, Z., Si, T., Liu, S.: Integration convolutional neural network for person re-identification in camera networks. IEEE Access **6**, 36887–36896 (2018)
95. Liu, Y., Song, N., Han, Y.: Multi-cue fusion: Discriminative enhancing for person re-identification. J. Vis. Commun. Image Represent. **58**, 46–52 (2019)
96. Wang, F., Zhang, C., Chen, S., Ying, G., Lv, J.: Engineering Hand-designed and Deeply-learned features for person Re-identification. Pattern Recogn. Lett. (2018)
97. Yuan, C., Guo, J., Feng, P., Zhao, Z., Xu, C., Wang, T., ... & Duan, K.: A jointly learned deep embedding for person re-identification. Neurocomputing **330**, 127–137 (2019)
98. Xin, X., Wang, J., Xie, R., Zhou, S., Huang, W., Zheng, N.: Semi-supervised person Re-Identification using multi-view clustering. Pattern Recogn. **88**, 285–297 (2019)
99. Wu, D., Zheng, S.J., Bao, W.Z., Zhang, X.P., Yuan, C.A., Huang, D.S.: A novel deep model with multi-loss and efficient training for person re-identification. Neurocomputing **324**, 69–75 (2019)
100. Zhou, S., Ke, M., Luo, P.: Multi-camera transfer GAN for person re-identification. J. Vis. Commun. Image Represent. (2019)
101. Zhong, W., Jiang, L., Zhang, T., Ji, J., Xiong, H.: Combining multilevel feature extraction and multi-loss learning for person re-identification. Neurocomputing (2019)
102. Fumera, B.L.G., Roli, F.: Multi-stage ranking approach for fast person re-identification. IET Comput. Vis. **12**(4), 513–519 (2018)

# Deep Learning in Gait Analysis for Security and Healthcare



Omar Costilla-Reyes, Ruben Vera-Rodriguez, Abdullah S. Alharthi,  
Syed U. Yunas and Krikor B. Ozanyan

**Abstract** Human motion is an important spatio-temporal pattern as it can be a powerful indicator of human well-being and identity. In particular, human gait offers a unique motion pattern of an individual. Gait refers to the study of locomotion in both humans and animals. It involves the coordination of several parts of the human body: the brain, the spinal cord, the nerves, muscles, bones, and also joints. Gait analysis has been studied for a variety of applications including healthcare, biometrics, sports, and many others. Until recently, the analysis has been done mainly by human observation, using parameters and features established in existing practice and therefore limited by the nature of measurements captured by the gait sensing modalities. In this chapter, we reviewed key conceptual and algorithmic facets of deep learning applied to gait analysis in two important contexts: security and healthcare.

**Keywords** Deep learning · Gait analysis · Biometrics · Dual-task · Machine learning

---

O. Costilla-Reyes (✉)  
Brain and Cognitive Sciences, Massachusetts Institute of Technology,  
77 Massachusetts Ave, Cambridge, MA 02139, USA  
e-mail: [costilla@mit.edu](mailto:costilla@mit.edu); [omar.costilla.reyes@gmail.com](mailto:omar.costilla.reyes@gmail.com)

R. Vera-Rodriguez  
Biometrics and Data Pattern Analytics (BiDA) Lab - ATVS,  
Universidad Autonoma de Madrid, Avda. Francisco Toms y Valiente, 11,  
28049 Madrid, Spain  
e-mail: [ruben.vera@uam.es](mailto:ruben.vera@uam.es)

A. S. Alharthi · S. U. Yunas · K. B. Ozanyan  
School of Electrical and Electronic Engineering, The University of Manchester,  
Manchester M13 9PL, UK  
e-mail: [abdullah.alharthi@postgrad.manchester.ac.uk](mailto:abdullah.alharthi@postgrad.manchester.ac.uk)

S. U. Yunas  
e-mail: [syed.yunas@manchester.ac.uk](mailto:syed.yunas@manchester.ac.uk)

K. B. Ozanyan  
e-mail: [k.ozanyan@manchester.ac.uk](mailto:k.ozanyan@manchester.ac.uk)

## 1 Introduction

This book chapter focused on understanding human motion behavior by modern machine learning techniques to solve complex, but interesting problems in the areas of security and healthcare. Specifically, the interest was to study human gait and footsteps, a periodic motion in time and space, and how this unique behavioral pattern can provide insights into applications such as detecting perturbation of gait under a cognitive task for wellbeing and biometric verification of individuals for security.

Gait, enables human motion, involves the coordinated interaction of many parts of the human body [1]. Consequently, gait is unique for every human being and is influenced by independent factors such as height, gender, fitness, and age. Gait can be expressed as a spatio-temporal pattern and offers value for a wide range of applications, ranging from biometric systems to the identification of markers of neurodegenerative diseases, such as Alzheimer's [2], based on long-term gait monitoring. Typically, both temporal and spatial parameters of gait, such as cadence, stride length, and others are analyzed from data acquired by a suitable floor sensor system or fused from several systems.

The first research theme of this book chapter focused on gait analysis for healthcare in the context of dual tasks. Gait has been shown to be affected by a managed cognitive load [3–6]. Walking while simultaneously performing a cognitive task, also known as 'dual-task' in literature, has been shown to induce walking variability in adults [3–5]. Moreover, in older adults, walking whilst performing a secondary task such as talking has been shown to increase the risk of falling [7]. Dual-task research aims to understand the relationship between cognitive activities and gait.

The negative impact of dual-tasks in participants with the neurodegenerative disease may be greater than in cognitively healthy adults [6]. This finding suggests that the cognitive capacity that each participant brings to the walking task may play an important role in the walking patterns. This finding has implications for finding a possible gait-related behavioral marker or 'biosignature' indicating the early stages of neurodegenerative diseases.

The second research theme of this book chapter was focused on a verification biometric problem applied to security [8]. Biometrics is an area that deals with the design of security systems for automatic identification or verification [9] of a human subject (client) based on physical and behavioral characteristics. Physical biometric traits include fingerprints, facial features, and iris. Behavioral biometrics, such as gait recognition, are intended to capture unique signatures delivered by client's natural behavioral patterns. This approach is useful since the complexity in reproducing such patterns by an impostor (intruder) is difficult. Biometric recognition by gait is based on the study of human locomotion to obtain a distinct biometric signature of a client. Twenty four unique factors have been shown to affect human gait [10], resulting in a singular gait pattern for every individual.

Moreover, a biometric system based on gait requires users to exert minimum effort from users for appraisal. A gait biometric system can be deployed in scenarios,

ranging from airport entry checkpoints and entry to buildings to home-based security systems. Feature engineering has been central in automatic gait recognition research [11]. The procedure involves the careful selection and design of complex and time-consuming hand-crafted features from footstep data, employing geometric, holistic, spectral and wavelet feature engineering approaches to name some [12].

For both themes, the research effort of this book chapter focused on designing machine learning models based on Convolutional Neural Networks (CNN), a form of deep learning [13], to allow the automatic extraction of features from the raw spatio-temporal gait and footstep data.

The ImageNet Large Scale Visual Recognition Challenge [14] is one of the largest computer vision competitions in the world. The challenge objective is to classify images from a 1000 set of possible labels such as “car”, “plane”, etc. The dataset contains around 1 million images. The breakthrough of deep learning in modern time came from using this massive dataset for image classification by using convolutional neural networks. The best accuracy results of the challenge in recent years (from 2014 onwards) have used convolutional neural network techniques at its core [14].

## 2 Gait Analysis Review

Gait analysis has been widely studied for a variety of applications including healthcare, biometrics, sports, and many more [1]. Classification of a person’s given its emotional state has also been explored. A person’s pride, happiness, neutral emotion, fear, and anger has been classified with high statistical confidence given only its gait pattern [15]. Generally, three types of gait monitoring systems exist, namely: cameras using image processing, floor sensors and wearable sensors [11]. The use of cameras for gait is vulnerable to details in the environment such as levels of lighting. Besides that, the use of cameras is considered an invasion of privacy in living environments, e.g. for healthcare [16]. Because of disadvantageous parallels to video surveillance. The disadvantage of wearable sensors is that the sensors need to be attached to the body, maybe uncomfortable to wear, as well as require assistance to attach correctly. On the other hand, floor sensor systems have the advantage of being non-invasive and even unobtrusive, less prone to environmental noise and undemanding the subject’s attention, which affects the data quality positively.

Gait occurs due to a cooperation of several parts of the human body including the brain, spinal cord, nerves, muscles, bones, and joints [1]. Within a walking sequence, gait can be understood as a translation of human brain activity to the patterns of muscle contractions. The command is generated by the human brain which is transmitted to initiate the neural centers through the spinal cord which eventually results in patterns of muscle contractions supported by the feedback from muscles, joints, and the receptors. This will result in the movement of the trunk and lower limbs in a connected way whilst the feet recursively touching ground surface and the change center-of-mass of the human body. Gait can be defined as repetitive cycles for each

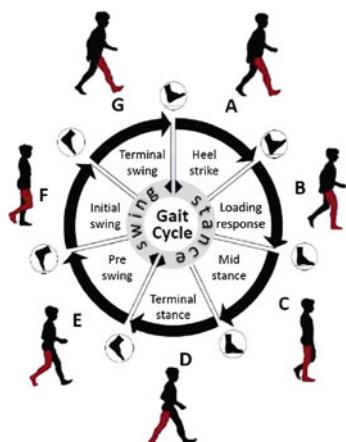
foot resulting in a sequence of periodic events. Each cycle can be divided into stance and swing phases as shown in Fig. 1.

The classification of a person's given its emotional state has been explored in the literature. A person's pride, happiness, neutral emotion, fear, and anger has been classified with high statistical confidence given only its gait pattern [15]. Generally, three types of gait monitoring systems exist, namely: cameras using image processing, floor sensors and wearable sensors [11]. The use of cameras for gait is vulnerable to the environment such as lighting. Besides that, the use of cameras invades the privacy in living environments, e.g. for healthcare [16] because of disadvantageous parallels to video surveillance. The disadvantage of wearable sensors is that the sensors need to be attached to the body, maybe uncomfortable to wear, as well as require assistance to attach them correctly. On the other hand, floor sensor systems have the advantage of being non-invasive and even unobtrusive, less prone to environmental noise and undemanding the subject's attention, which has a positive effect on data quality.

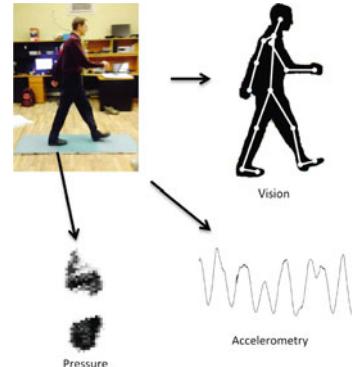
Within a walking sequence, gait can be understood as a translation of human brain activity, projected into the spinal cord and then able to activate the patterns of muscle contractions. The command is generated by the human brain which is transmitted to initiate the neural centers through the spinal cord which eventually results in patterns of muscle contractions supported by the feedback from muscles, joints, and receptors. This results in movement of the trunk and lower limbs in a connected way whilst the feet recursively touch the ground surface and the change center-of-mass of the human body. Gait can be defined as repetitive cycles for each foot resulting in a sequence of periodic events. Each cycle can be divided in phases shown in Fig. 1, defined as follows:

- Stance Phase (approximately 60% of the gait cycle, with the foot in contact with the ground). This phase is subdivided into four intervals (A, B, C, D).

**Fig. 1** Gait cycle [17]



**Fig. 2** Modalities of gait analysis. Including vision, pressure and accelerometers [18]



- Swing Phase (approximately 40% of the gait cycle with the foot swinging and not in contact with the ground). This phase is subdivided into three intervals (E, F, G).

The three main modalities to study gait is by image processing, floor sensors and with wearable devices. The modalities are shown in Fig. 2 [18]. Gait patterns can be obtained from video streams, floor sensor systems footstep pressure or with accelerometer signals (temporal signals).

Gait analysis in the context of this work deals with two main components. One component is time: this refers to the temporal gait cycle pattern. The other component is space: this refers to the spatial footstep shape characteristics of the gait pattern. Here, we introduce a methodology to learn spatio-temporal features directly from raw sensor data with deep learning models, this is without the use of human feature engineering. The deep learning models are based on ANN architectures of several layers that are able to learn features automatically from raw sensor data.

## 2.1 Non-wearable Sensors

**Camera-based sensors.** Here images and videos obtained from cameras record human gait. Then, image processing techniques are used, such as segmentation and others to identify gait from the images. This is the most widely used approach in the literature. This approach involves both model-free and model-based analysis for gait recognition [12].

In Table 1 are shown the state-of-the-art research approaches for vision systems in biometrics applications. The performance is indicated as classification rate (CR). Histogram-based systems have been found to be the most successful in this area.

**Floor sensors.** The main research themes developed in this work are based on floor sensor systems. There are mainly two types of floor sensors studied in the literature based on the ground reacting force and the other based on switch sensors. The first obtains continuous signals while the latter only delivers binary pressure signals. In

**Table 1** State-of-the-art vision systems for gait analysis in biometrics

Feature set	Performance	Subjects
Histogram and silhouette [19]	97.5% CR	3141
Histogram and deep silhouette [20]	97.7% CR	176
Histogram and energy image [21]	79.8% CR	122
Chrono-gait image [22]	51.3% CR	122
Gait energy volumes [23]	95% CR	25

**Table 2** State-of-the-art floor sensor systems for gait analysis in biometrics

Research group	Signals	Users	Samples	Model	Multi-shoe	Norm.	Results (EER)
Cattin [26]	470	16	6 step cycles	Euclidean distance	Yes	No	9.45
Stevenson et al. [27]	88	8	85 step cycles	HMM	Yes	Yes	20
Mostayed et al. [28]	18	6	5 step cycles	Histogram similarity	No	No	3.3 to 16
Vera-Rodriguez et al. [29]	9900	5	500 step cycles	SVM	Yes	Yes	2.6 val, 4 eval
Mason et al. [12]	399	10	30 step cycles	LDA	Yes	Yes	1.52 val, 3.1 eval
Costilla Reyes et al. [8]	9900	127	500 step cycles	Resnet and SVM	Yes	Yes	0.7 val, 1.70 eval

Table 2 are shown the state-of-the-art footstep recognition systems. The table shows the number of signals, the number of users, samples type per model and performance results in EER.

The iMagimat [24, 25] is an affordable floor sensor that allows spatio-temporal sampling of the ground reaction force (GRF) resulting from footsteps. The gait data can be recorded, stored and analyzed over large periods of time. The technology is embodied in a 1m by 2m prototype [24]. Gait is measured by detecting light attenuation caused by the bending of plastic optical fibers (POFs) while walking on the surface. GRF in the active area of the sensor can then be reconstructed for further data analysis [24, 25]. The adequate spatio-temporal sampling is ensured by applying tomography principles to the floor sensor design and a suitable frequency of spatial frames acquisition set at 256Hz.

### 2.1.1 Advantages

- Multiple gait parameters can be obtained from a wide set of modalities
- There are no power constraints limitations

- Non-intrusive system
- There are no external factors affecting the analysis due to a controlled environment.

### 2.1.2 Disadvantages

- Area of measurement is confined to a limited space
- Sensor system is often expensive
- May allow gathering signals without the users' consent or knowledge
- Not suitable for outdoor applications

## 2.2 Wearable Sensors

In this approach, sensors are placed in different positions of the human body to measure gait. The type of sensors that can be used are force sensors, accelerometers, gyroscopes, extensometers, and inclinometers.

**Inertial sensors.** These sensors use the earth's gravitational field to obtain measurements of a subject velocity, acceleration, orientation or gravitational forces for gait analysis. Three-axis accelerometers and gyroscopes angular velocity is usually used for this type of application.

In Table 3 are shown the state-of-the-art approaches for inertial systems. There is currently no consensus from the research community in approach or location of the sensor for optimal analysis.

**Ultrasonic sensors.** Sound waves are used as the sensing mechanism. By measuring the distance between the ultrasonic sensor and the progression of the gait pattern, gait can be measured and consequently studied.

**Electro goniometer.** This sensor, often installed in the hip or knee allows the obtention of continuous measurements of the current states of a joint angle of a human subject.

**Table 3** State-of-the-art inertial systems for gait analysis in biometrics

Feature set	Performance in EER %	Subjects	Location of sensing
Inner product of acceleration [30]	6.8	740	Pocket
Key points of acceleration [31]	2.2	175	5 locations
Gait cycle acceleration [32]	1.6	60	Hip
Direct matching [33]	1.6–6.1	30	Ankle
Dynamic time warping [34]	5.6	21	Spine

**Exoskeletons.** They are devices that cover the entire human body, usually made of solid materials. These devices are combined with goniometers or potentiometers sensors to allow measurement of human kinematics.

### 2.2.1 Advantages

- Long-term gait analysis is allowed
- System may be inexpensive
- Controlled environments are not necessary, therefore natural gait signals may be obtained
- Suitable for outdoor applications
- Freedom to focus the analysis in different locations of the human body

### 2.2.2 Disadvantages

- Power consumption limitations
- Due to portability, only a limited set of gait parameters can be studied
- susceptible to noise and interference of external environments.

In summary, the floor sensor system offers unique advantages over other sensing modalities to analyze gait. The system is non-intrusive and resilient to noise in the environment as the main advantages over other sensing systems. As opposed to forcing the user to wear the device for the experiment as in some inertial systems or to be susceptible to noise in environmental conditions such as different levels of light or cross-view angles difficulties to acquire the data in a form suitable for analysis [35]. For the aforementioned reasons, this book chapter focuses on studying the ground reacting force from floor sensor systems for gait analysis in two applications healthcare and security.

## 2.3 A Review of Floor Sensor Systems and Datasets for Gait Analysis

Cameras, inertial sensors or floor sensor systems have been used for gait analysis [11, 36]. Floor sensor systems have the advantage of being unobtrusive and resistant to surrounding noise; in contrast, camera systems require adequate illumination while wearable inertial sensors require daily placement and maintenance. A floor sensor system can be hidden in a home environment allowing the acquisition of natural gait signals over large periods of time. While floor sensor systems have been built for automatic gait analysis applications [11], they have relied heavily on physiologically defined, man-made features such as the body's center of pressure, stride length, and cadence, rather than using raw sensor signals, to construct gait classifier models.

An example of gait recognition system using a switch sensor system is the UbiFloorII system [37]. The switches in the UbiFloorII system are made from photo interrupters sensors. The switch sensor generates 0 V or 5 V (on-off) according to the weight exerted on the floor sensor system.

Force plates [38] have been used as the sensors used for gait analysis to obtain the ground reaction force, perpendicular to the floor sensor system. Piezoelectric sensors are used as the sensing mechanism. The piezoelectric effect measures the accumulated charge in solid materials as a response to force stress. In this case, the measured pressure is the response to the pressure exerted by the weight of the subject walking on the floor. The change in pressure modifies the voltage level in the piezoelectric sensor output, to enable the measurement of gait signals.

For the goal of classification of human postural and gestural movements using floor sensor systems, Saripalle et al. [39] applied force platforms to infer the center of pressure of individuals. Eleven body movements by volunteers were analyzed with an accuracy ranging from 79 to 92% using linear and non-linear supervised machine learning models. Feature selection is highlighted as a critical step for obtaining reliable accuracy scores, but this approach is limited by the lack of a single classification model suitable for all types of mobility.

Floor sensors systems have been used to distinguish human movements as presented in [40]. The recognition is achieved by analyzing the Ground Reaction Force (GRF) on a weight-sensitive floor. The changes in the GRF arise from activities performed at the same position, including jumping, sitting and rising. A hidden Markov model was used for human movement classification. The classification performance was close to 100%. One of the disadvantages of such a study is that the postural activities were performed statically at the same position.

### 3 Deep Learning for Gait Analysis

Supervised machine learning is a category of artificial intelligence (AI) and a specific kind of machine learning. Algorithms or mathematical models are built and trained with a given set of inputs and desired outputs. The models are tested on unseen data by exploring the structure of the data and fit into the models which can be understood and utilized by the users [41]. Shallow Learning depends on handcrafted features learned in a predefined relationship between the inputs to the output: such as linear regression, logistic regression, decision tree, Support Vector Machine (SVM), random forest, naive Bayes, and k-nearest neighbor.

Supervised learning is the most widely used technique in the industry for classification and prediction. For example, Google Inc. uses supervised learning to classify the email as spam or not spam or to rank web pages for their search engine by using the page rank [42] algorithm, Facebook Inc. uses supervised learning to automatically tag people in pictures uploaded to the social site [43]. Amazon Inc. uses supervised learning as a recommender system [44] to buy products based on user history. Those

examples are just a small subset of the wide applications of supervised learning in industry.

Deep structured learning or hierarchical learning is inspired by the biological neural networks structure and function. It is based initially on the concept of multi-layer Artificial Neural Network (ANN) with the aim to learn data representations automatically; thus, Deep Learning becomes the method of choice where the classification features, if known at all, are complex, with no straightforward quantitative relation to the raw data. Typically, the term ‘deep’ refers to the number of layers in the variety of possible networks structures: Deep Belief Networks (DBN), Feed-forward Deep Networks (FDN), Boltzmann Machine (BM), Generative Adversarial Networks (GAN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long-Short Term Memory (LSTM) a special kind of RNN. A comprehensive presentation of the theory of ANNs and Deep Learning is not within the scope of this review and the reader is referred to established sources [13]. Further, we focus on models with practical significance for gait applications such as CNN and LSTM.

The CNN model is suitable for processing 1D, 2D or 3D data that has a known grid-like topology [13]. The network has the ability to learn a high level of abstraction and features from large datasets by applying convolution operation to the input data. Commonly, the network consists of convolution layers, pooling layers, and normalization layers, with a set of filters and weights shared among these layers.

The convolutional layers output a feature map harvested automatically from the raw input data. The pooling layers are utilized to reduce the size of representation and make the convolution layer spatially invariant. The CNN model uses commonly two types of pooling layers: max pooling and average pooling. All convolution layers and pooling layers have activation functions (e.g. Sigmoid, Tanh, ReLU, Leaky ReLU), to calculate the weight of neuron and add a bias, deciding whether to fire the neuron or not [45]. LSTM networks are favorable for processing time-series data, where the order is of importance, such as gait data sequences. In essence, they exploit recurrence, by using information from a previous forward pass over the network.

The goal of using ANNs in gait analysis is to develop a model to extract gait features and perform well on unseen real-world gait data. Commonly, for appropriate training and testing, the model is trained and validated on 70% of the data and tested on the remaining 30%. In supervised training, the procedure is launched by initializing the weights randomly, processing the inputs and comparing the resultant output against the desired output. During training, the weights and biases are adjusted in every iteration, until the error is minimized, and validation is used to estimate the model performance during training. Lastly, the model is tested with unseen data, allowing to identify over-training.

The widely used accuracy measure for ANN gait analysis is the confusion matrix. It is a table to visualize the number of predictions classified correctly and wrongly for each class. The table consists of true positive, true negative, false positive, and false-negative classification occurrences. One of the advantages of the confusion matrix display is that it is straightforward to identify the decision confusions, thus possibly concluding on the quality of the data involved.

### 3.1 Convolutional Neural Networks

#### 3.1.1 Convolutional Layer

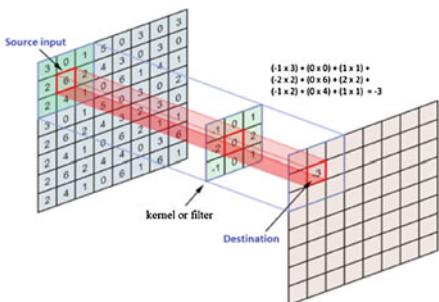
A core building block in CNN models is the convolutional Layer where the computational heavy lifting of data processing is taking place. This layer is based on Convolution, a specialized kind of linear operation [30]. The convolution operation is performed on two functions to produce a feature map, where the first function is the input data and the later is the filter or kernel. In this process, the filter slides over an input data and perform convolution, the sum of the convolution operations transformed to feature maps (see Fig. 3). Feature map output consists of different feature maps produced by different kernels as convolution layer output. An activation function is utilized to produce nonlinear feature maps to make the training faster and more accurate. The widely used activation function in a convolutional layer is Rectified Linear Units (ReLU) to convert all negative numbers to 0 or positive. A mathematical representation of convolution operation given an input  $I(t)$  and a kernel  $K(a)$  is given as

$$s(t) = \sum \alpha^N I(a) * K(t - a) \quad (1)$$

#### 3.1.2 Pooling Layer

There are usually two types of layers in convolutional networks, pooling layers, max pooling, and average pooling. The objective of using this layer is to recombine the convolutional layer output to produce meaningful information. In pooling layer, a filter slides over the convolutional layer output and the maximum or average value in the filter window are transformed as an element in an output matrix as pooling layer output.

**Fig. 3** Convolutional layer processing data to produce feature map



### 3.1.3 Training and Testing

In gait analysis, the goal of using neural networks is to develop a model to extract gait features and perform well on unseen real-world gait data. For appropriate training and testing, commonly the model is trained and validated on 70–80% of the data and tested on the remaining 30–20%. In supervised training, the procedure initializes the weights randomly, processing the inputs and comparing the resultant output against the desired output. During training, the weights and biases are adjusted in every iteration until the error is minimized, and validation is used to give an estimate of the model performance during training. Lastly, the model is tested with unseen data, allowing to identify over-fitting.

### 3.1.4 Convolutional Neural Networks for Spatio-Temporal Analysis

Recognize human actions from videos is an important spatio-temporal problem [46]. In recent years, the top-performing models to solve this problem have been based on CNN's and Recurrent Neural Networks [46–48].

The architectures use publicly available video datasets. These approaches are effective to learn representations from raw video frames. However, they are complex and require large computational resources to train; furthermore, in some cases, other pre-processing steps are required, such as to calculate optical flow between video frames [46]. Convolutional neural networks have been proposed to study spatio-temporal human recognition [49]. Including 3D convolution operations to capture both the spatial and temporal domain components [50]. Action scene understanding has also been proposed [51]. The two-stream convolutional network has shown to be effective for the spatio-temporal action recognition problems [46, 49].

The two-stream deep learning architecture [46, 49] utilizes an end-to-end learning approach for analyzing the spatial and temporal streams of videos in two separate deep networks. The spatio-temporal information is combined at a feature or score level after the last layers in the network. However, this approach sometimes involves computationally heavy calculations such as optical flow [46].

Gait analysis from the video (spatio-temporal features) has been widely studied in the literature [12, 35, 52]. Wu et al. [35] presented a study of cross-view gait for human identification, using deep convolutional neural networks models in three gait datasets. The results show a substantial increase in the average recognition rate performance when compared with the previous state-of-the-art. For example, in the CASIA-B dataset, the average recognition rate reaches 94.1% with a deep network. This compares favorably with the previous best recognition rate of 65% by hand-made feature engineering.

## 4 Deep Learning in Healthcare: A Case Study in Dual-Tasks

We investigated cognitively demanding tasks on patterns of human gait in healthy adults with a deep learning methodology that learns from raw gait data. Age-related differences were analyzed in dual-tasks in a cohort of 69 cognitively healthy adults organized in stratified groups. A novel spatio-temporal deep learning methodology was introduced to effectively classify dual-tasks from spatio-temporal raw gait data, obtained from a tomography floor sensor. The approach outperformed traditional machine learning approaches. The most favorable classification F-score obtained was of 97.3% in dual-tasks in a young age group experiment. The deep machine learning methodology outperformed classical machine learning methodologies by 63.5% in the most favorable case. Finally, a 2D manifold representation was obtained from trained deep learning models' data, to visualize and identify clusters from features learned by the deep learning models. Here, we demonstrate a novel approach to dual-task research by proposing a data-driven methodology with stratified age-groups.

### 4.1 Aims and General Method

This study aims to establish a benchmark in a relationship between a managed cognitive load and gait in cognitively healthy participants from a data-driven analysis perspective, in contrast to the traditional studies found in the gait literature, that take advantage of gait parameters such as gait speed variability, walking base and others [3–5, 53]. This study proposes a novel analytic approach to gait analysis based on advanced computational models known as deep machine learning [13].

Current methods in dual-task analysis rely on specific statistical features such as gait speed and variability [3–5, 53]. These studies rely on a limited number of features and parameters to make inferences about the gait patterns. Here, we use a data-driven approach to learn the parameters. Moreover, sometimes few experimental gait samples per participant have been included in the analysis in traditional approaches.

In this study, the dual-task effects are studied using deep machine learning principles [13] to automatically define and extract most favorable features harvested from raw spatio-temporal gait data and selected by the model. The data were obtained from an original tomographic floor sensor system base on sensing enabled by plastic optical fibers [25] sampled directly from the raw sensor data rather than from reconstructed data [24], which requires further processing.

A large cohort of 69 participants participated in the study. They performed the tasks for 5 min. The approach resulted in a sizeable set of gait samples per participant experiment that allowed statistical reliability [54]. Furthermore, large datasets are beneficial for the most favorable application of deep learning models.

## 4.2 Background

In dual-tasks, a participant performs a cognitive task whilst walking [4] to measure the effect of the cognitive task on walking patterns. Studies have shown that stride velocity changes and gait variability increases during the performance of a cognitively demanding task [3–5]. During dual-tasks a participant's executive function, postural control and the ability to walk are altered. Dual-tasks have caused pronounced changes in gait from mild-cognitive impaired (MCI) participants when compared to healthy control groups [53].

Several factors may influence changes in gait while performing cognitive tasks. For example, the impact of a cognitive task on walking speed has been linked to the difficulty of the task and on the nature of the walk. Other factors that might influence dual-tasks include anxiety, happiness, and other emotional states, which have received less attention in the gait analysis literature [55]. Moreover, a clear consensus on the effect of cognitive load on walking patterns has yet to emerge in the literature [4].

## 4.3 Methodology

### 4.3.1 Inclusion Criteria

Healthy men and women between the ages of 20 and 65 years were invited to participate in the study. Those with any condition that might affect a normal walking pattern, typically a history of falls within 6 months prior to enrolment, were excluded from the study. Statistical information such as gender and age were also captured to allow further analyses. All methods were performed in accordance with guidelines and regulations by the University of Manchester Research Ethics' Committee. Informed consent was also obtained from the participants to take part in this study.

Statistical information such as gender and age were also captured to allow further analyses. All methods were performed in accordance with guidelines and regulations by the University Research Ethics Committee (UREC) at the University of Manchester. The experimental protocol was approved by the Ethics' Committee with reference: *ethics/15536* on January 25, 2016. Informed consent was obtained from the participants to take part in this study.

### 4.3.2 Procedure

Four walking experiments were executed by each participant on the floor sensor system. The participants initially undertook normal and fast walk experiments. Followed by two dual-task experiments. The first dual-task experiment was to spell five common words in reverse [3]. In the second, participants performed serial seven

subtractions starting from a random 3-digit number [5]. The experiments were performed in a silent environment, with external distraction kept to a minimum. Participants were allowed to wear any type of footwear during the experiments. Each experiment lasted 5 min for a total of 20 min per participant. The number of captured experimental gait samples depended on the participant's speed and manner of walking, which varied among participants. The participants walked continuously from one end of the walkway to the other during the experiment. An extra one-meter length at the start and end of the floor sensor system was allowed to enable the participants to accelerate and decelerate their walk. No cameras or video recording was used since they can significantly compromise the privacy of participants [16] and affect adversely the quality of the data.

#### 4.3.3 Description of Experiments

1. *Normal walk experimental task*: Participants walked at normal self-selected speed for the duration of the experiment.
2. *Fast walk experimental task*: Participants walked at self-selected fast walking speed.
3. *Dual-task one, reverse spelling experimental task*: Participants were given a set of five-letter common words and they had to spell the word continuously backward out loud [3] for the duration of the experiment. For example, spell “earth” or “could”.
4. *Dual-task two, backward serial subtraction experimental task*: Participants subtracted seven from a random three-digit number continuously out loud during the five minutes of the experiment [5].

#### 4.3.4 Database: UoM-Gait-69

The dual-task database collected, entitled UoM-Gait-69, was comprised of data from 69 cognitively and physically healthy adults who participated in our study. The participant's ages ranged from 20 to 63 years. Thirty-seven (53%) were female. The participants were given a unique identification number (ID) for anonymization and experiment identification.

### 4.4 Experiments for Age-Related Classification

We designed a set of seven experimental cases (rows of Table 4). The cases are the database volunteers arranged in different groups to allow experimental results. For example, Experimental task 1 has 3 groups: group 1 of 27 participants between 20–28 years, group 2 of 22 participants between 31–42 years and group 3 of 20 participants between 46–63 years. Experimental cases Table 4.

The experimental cases have a wide range of age-cohort sets (group columns of Table 4) of age-related differences in dual-tasks executed by the participants with the aim of testing the ability of a machine learning model to differentiate age-range sets with a variable number of participants per set. This includes experiment type, number of participants and age range. In some experiments, a large age cohort of participants was contained in each group. For example, experiment one had three-decade-long age sets for classification of approximately 20 participants in each set. Also, a single age cohort was included in other experiments such as experiment seven, for participants between 20 and 26 years old of age.

### ***Spatio-Temporal Raw Sensor Matrices***

Spatio-temporal raw sensor matrices (RSMs) as described in [25] were constructed from the raw sensor data in this study. This approach did not require tomography reconstructed images, instead, it was possible to derive the RSMs directly from the raw data. Therefore, RSMs were calculated for all the experiments performed for this study.

## ***4.5 Spatio-Temporal Deep Learning Model***

The deep learning inception architecture, shown in Fig. 4, contains two-stream Inception-like modules that model space and time within the same deep network. One of the streams was assigned to the temporal domain whilst the other to the spatial domain. The spatio-temporal streams were optimized at the same time by back-propagation [13]. Each stream ends before a fully connected layer of 100 neurons to allow equilibrated spatio-temporal feature concatenation. Then the outputs of the network were passed through a last fully connected layer with softmax activation that allowed gait sample classification. After this layer, classification performance results were obtained according to the experiment type (see Table 4). ReLU activations are used in all layers [13].

**Table 4** Description of 7 experimental tasks

Experiment	Group 1		Group 2		Group 3	
	Age	Participants	Age	Participants	Age	Participants
1	20–28	27	31–42	22	46–63	20
2	20–21	8	–	–	56–63	8
3	20–28	27	–	–	46–63	20
4	20	6	30–32	6	60–63	6
5	20	6	–	–	60–63	6
6	20	6	35–36	6	60–63	6
7	20	6	26	6	–	–

**Fig. 4** Deep learning spatio-temporal architecture (inception inspired). Temporal and spatial streams are concatenated at the feature level before a last fully connected layer to obtain classification scores

#### 4.6 Results for Age-Related Differences

Classification experiments were performed in three age groups organized in 7 experiments. As shown in Table 4, age group one, the youngest age set, ranged from 20 to 28 years. Age group two, the middle age set group, ranged from 31 to 42 years (with the exception of experiment seven, a 26-year-old only group). Age group three, the older age group, contained participants with age ranges between 46–63 years.

The classification results are reported as a triple measure, containing F-score percentage performance for normal walk compared to (1) fast walk (2) dual-task one and (3) dual-task two.

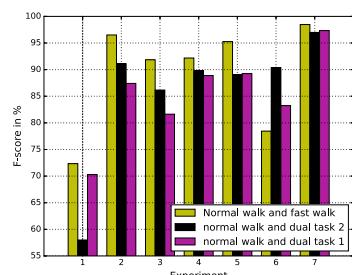
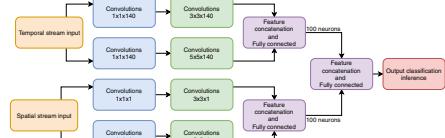
The highest classification performance model was obtained for experiment 7, whilst the lowest performance was obtained for experiment 1. In the former two age groups containing a single age were classified while the latter had the largest number of participants distributed by age group. These characteristics influenced the classification performance. In Fig. 5 is shown F-score performance summary of all the experiments.

Experiment 7 contains 6 participants per single age (ages 20 and 26 years). While experiment 3 contains a large number of participants in wide age ranges. 27 participants between the ages of 20–28 years and 20 participants between 46–63 years. The wide age ranges found in experiment 3 offer a higher challenge to the classification algorithms to discriminate the gait samples correctly, this is in contrast with experiment 7 which contains a single age (Table 4).

#### Model comparison

Ensemble and linear machine learning models were also implemented to compare with the results presented in Sect. 4.3. Table 5 demonstrates that the Random Forest

**Fig. 5** F-score performance summary of the seven age-related experiments. Y axis: F-score in % X axis: number of experiment



**Table 5** Age-related differences classification results

Experiment	Classification	Total classes	Age groups	Precision (%)	Recall (%)	F-score (%)	Support
1	Normal and fast walk	6	3	81.21	74.53	72.34	7052
2		4	2	96.6	96.51	96.51	2152
3		4	2	91.93	91.87	91.85	4832
4		6	3	92.62	92.38	92.18	1889
5		4	2	95.39	95.29	95.26	1316
6		6	3	82.52	79.78	78.45	1785
7		4	2	98.48	98.47	98.47	1377
1	Normal walk and dual-task one	6	3	71.29	70.48	70.28	6365
2		4	2	87.49	87.46	87.41	1922
3		4	2	81.7	81.64	81.63	4340
4		6	3	88.93	88.96	88.89	1712
5		4	2	89.5	89.23	89.25	1189
6		6	3	84.74	83.94	83.24	1874
7		4	2	97.33	97.33	97.33	1236
1	Normal walk and dual-task two	6	3	57.33	58	58	6254
2		4	2	91.18	91.13	91.11	1848
3		4	2	86.4	86.13	86.16	4262
4		6	3	89.77	89.79	89.76	1655
5		4	2	89.19	89.09	89.06	1118
6		6	3	91.26	90.58	90.37	2092
7		4	2	97	96.99	96.99	1197

returned an F-score of 56.12% whilst the linear SVM returned the lowest classification performance overall with an F-score of 23.67%. The deep learning methodology (F-score: 97%) improved the F-score of the Random Forest classifier by 40.88%, while the best improvement was obtained against the linear SVM classifier by 63.5%. These results justify a conclusion of robust classification performance of the deep machine learning methodology compared to a shallow, ensemble and linear machine learning models (Table 6).

**Table 6** Comparison of the deep learning models against shallow models of experiment seven. Classification of normal and dual-task two is shown. Classes are defined in Table 5

Optimization	Model	Precision (%)	Recall (%)	F-score (%)	Support
Early stop	Two-stream inception	97	96.99	96.99	1197
Genetic programming	Gradient boosting classifier	62.79	62.82	62.50	1197
None	Random forest	57.28	57.31	56.12	1197
None	Linear SVM classifier	26.36	27.82	23.67	1197

## 4.7 Analysis of Experiments Three and Seven

Here, experiment three and seven, described in Table 4 are further explored, since the former had a large cohort of participants in two age-ranges, whilst the latter delivered the best F-score overall experiments, for classification of two single-age groups. Tables 7 and 8 show the detailed performance results per class for experiment 3 and 7 respectively. The analysis included metrics such as the Matthews correlation coefficient, informedness, markedness, and prevalence [56] to further inform the classification performance results.

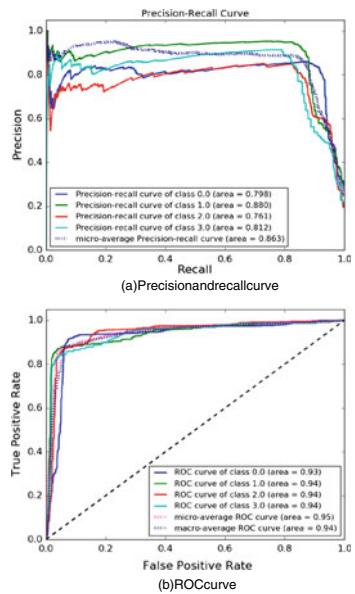
Figures 6a and 7a show the precision and recall curve [56] for experiments three and seven respectively that plots precision and recall correspondence for some threshold values. In Figs. 6b and 7b it is shown the receiver operating characteristic curve (ROC) [56] for the same two experiments. This curve demonstrates true positive and false positive threshold rates of a machine learning model. As in the case of the precision and recall curve, the experiment seven model outperforms experiment three in the ROC curve.

## 4.8 Discussion

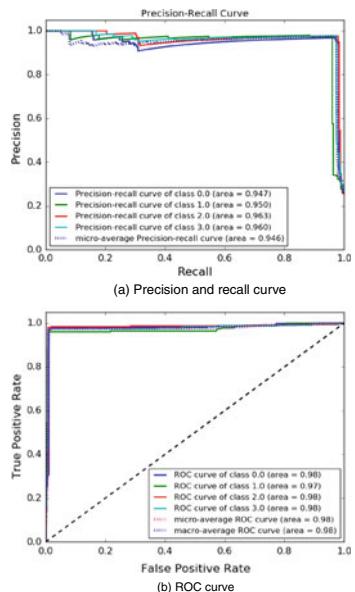
Our results demonstrated age-related differences in gait patterns in 7 experimental cases from a large cohort of healthy adult participants. The experiments compared normal walk, fast walk and two managed cognitive load activities. The spatio-temporal problem was addressed by a deep machine learning methodology which has the ability to learn end-to-end from raw spatio-temporal gait data. The effectiveness of the approach was justified by comparing our results to the performance of optimized shallow machine learning models' in linear and ensemble machine learning models.

Overall, the optimal performance of the methodology was observed between normal walk and fast-walk. Then between normal walk and dual-task two, and finally between normal walk and dual-task one. Task two influenced a heavier cognitive

**Fig. 6** Classification performance characteristics of experiment three



**Fig. 7** Classification performance characteristics of experiment seven



**Table 7** Performance per class of experiment three

Experiment	Experiment 3: Normal and dual-task two			
Classes	0	1	2	3
Population	4262	4262	4262	4262
P: Condition positive	1251	1279	833	899
N: Condition negative	3011	2983	3429	3363
Test outcome positive	1362	1187	876	837
Test outcome negative	2900	3075	3386	3425
TP: True Positive	1137	1096	710	734
TN: True Negative	2786	2892	3263	3260
FP: False Positive	225	91	166	103
FN: False Negative	114	183	123	165
TPR: Sensitivity, hit rate, recall	0.91	0.86	0.85	0.82
TNR = SPC: Specificity	0.93	0.97	0.95	0.97
PPV: Pos Pred Value (Precision)	0.83	0.92	0.81	0.88
NPV: Neg Pred Value	0.96	0.94	0.96	0.95
FPR: False-out	0.07	0.03	0.05	0.03
FDR: False Discovery Rate	0.17	0.08	0.19	0.12
FNR: Miss Rate	0.09	0.14	0.15	0.18
MCC: Matthews correlation coefficient	0.81	0.84	0.79	0.81
Informedness	0.83	0.83	0.8	0.79
Markedness	0.8	0.86	0.77	0.83
Prevalence	0.29	0.3	0.2	0.21
Positive likelihood ratio	12.16	28.09	17.61	26.66
Negative likelihood ratio	0.1	0.15	0.16	0.19
Diagnostic odds ratio	123.5	190.33	113.47	140.8
False omission rate	0.04	0.06	0.04	0.05

load in participants compared to task one, resulting in a more pronounced gait pattern, which impacted on the ability of the machine learning model to classify gait patterns successfully. Moreover, for participants to perform the arithmetic operations of dual-task, coordination among several processes such as articulatory, phonatory and respiratory, functions was required, which, might have led to a greater demand on the executive function processing [5]. High classification performance was also observed with short age-range groups and with a large age gap between groups. These characteristics tended to isolate the gait pattern even further.

The high classification performance obtained in the age-related experiments demonstrated that the deep learning methodology presented here may be appropriate for gait data analysis from participants with MCI in large cohort studies [57]. People with impaired executive function in the context of a diagnosis of AD have

**Table 8** Performance per class of experiments seven

Experiment	Experiment 7: Normal and dual-task two			
Classes	0	1	2	3
Population	1197	1197	1197	1197
P: Condition positive	288	280	308	321
N: Condition negative	909	917	889	876
Test outcome positive	292	273	309	323
Test outcome negative	905	924	888	874
TP: True Positive	281	267	301	316
TN: True Negative	898	911	881	869
FP: False Positive	11	6	8	7
FN: False Negative	7	13	7	5
TPR: (Sensitivity, hit rate, recall)	0.98	0.95	0.98	0.98
TNR = SPC: (Specificity)	0.99	0.99	0.99	0.99
PPV: Pos Pred Value (Precision)	0.96	0.98	0.97	0.98
NPV: Neg Pred Value	0.99	0.99	0.99	0.99
FPR: False-out	0.01	0.01	0.01	0.01
FDR: False Discovery Rate	0.04	0.02	0.03	0.02
FNR: Miss Rate	0.02	0.05	0.02	0.02
MCC: Matthews correlation coefficient	0.96	0.96	0.97	0.97
Informedness	0.96	0.95	0.97	0.98
Markedness	0.95	0.96	0.97	0.97
Prevalence	0.24	0.23	0.26	0.27
LR+: Positive likelihood ratio	80.63	145.74	108.6	123.19
LR-: Negative likelihood ratio	0.02	0.05	0.02	0.02
DOR: Diagnostic odds ratio	3277.12	3118.42	4735.38	7845.83
FOR: False omission rate	0.01	0.01	0.01	0.01

shown an exaggerated dual-task effect in walking patterns compared to cognitively healthy controls [58]. This finding has yet to be verified using deep machine modeling applied to a large sample of participants who have prodromal AD. This might enable finding a robust behavioral marker of AD in the very early prodromal stage.

#### 4.9 Future Directions

The methodology presented here has the potential to be expanded to a multimodal sensor fusion in a set-up for ambient sensing of natural human behavior. Holistic

analysis of an unobtrusively captured combination of voice patterns, upper body movement, gaze and other human behavioral patterns of participants. The approach has the potential to improve our understanding of their relationship in human behavioral pattern understanding to cognitive decline to the benefit of the detection of neurodegenerative diseases in the prodromal stage.

## 5 Deep Learning in Security: A Case Study in Biometrics

The advantage of gait as a biometric modality is that it allows detection at a distance, in an unobtrusive environment, it is inexpensive, and difficult to be forged by an intruder. However, gait is not a perfect biometric marker, since it has also raised privacy concerns. Users of biometrics systems have raised concerns in regards to the intrusive nature of the system, due to its ability to acquire signals without the users' consent or knowledge. The difficulties in considering gait as a biometric include change in client's clothes, shoes, subject emotions, among other factors. This is where machine learning methods for gait analysis have shown to be effective.

Footstep recognition uses force signatures made by person footsteps over a floor sensor system. This force is known as Ground Reaction Force (GRF). In contrast to obtaining gait by video streams, it's non-intrusive and less prone to effects in environmental noisy conditions that might diminish the performance of the system.

Footstep patterns tend to contain a high degree of variability, thus making visual assessment difficult. The discovery of a system that is able to robustly find and isolate these patterns automatically is at the forefront of gait biometric research.

Footstep data was collected from 127 users within an 18-month period. Three datasets were introduced for the experiments performed. Benchmark 1 (B1) dataset considers 40 stride footstep samples for 40 clients for training the machine learning models. This is the smallest dataset considered (in the number of available training signals per user), thus representing a security application. Benchmark 2 (B2) dataset considers 200 stride footstep samples for 15 clients for training, this represents a middle-level amount of footstep data available to train a machine learning model. An application might be for example at a supermarket or a workspace. Benchmark 3 (B3) dataset contains 500 stride footstep signals for 5 clients. This is the largest dataset considered, for an application where a large number of footstep signal can be acquired for training, as for example in a home-based scenario. In all cases, an evaluation dataset of 500 signals was considered.

### 5.1 Aims and General Method

This study aims to establish a benchmark for the relationship between a managed cognitive load and gait in cognitively healthy participants from a novel data analysis perspective. We will apply a novel analytic approach based on advanced computational

models known as deep machine learning [13]. Current methods in dual-task analysis rely on specific statistical features such as gait speed and variability [3–5, 50]. This focus has been influenced by human observation, which is intrinsically subjective and has limited reach. Usually, only a few experimental gait samples per participant have been included in the analysis.

In contrast, in this study, the dual-task effects are studied using deep machine learning principles [13] to automatically define and extract optimal gait features harvested from raw spatio-temporal gait data. The data were obtained from an original tomographic floor sensor system [51] sampled directly from the raw sensor data rather than from reconstructed data [52], which requires further processing. A large cohort of 69 participants was recruited, resulting in a sizeable set of gait samples per participant experiment that allowed statistical reliability [53]. These aspects are features not commonly found in gait analysis research. Furthermore, a large dataset is beneficial for the optimal application of deep learning models." as provided in the latex source code.

## 5.2 *Footstep Data as a Biometric*

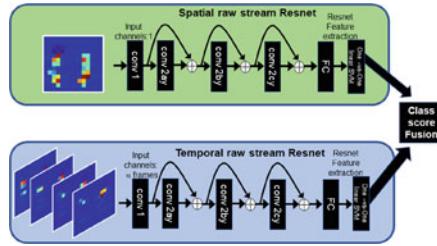
Footstep feature extraction and feature engineering have played a central role in automatic footstep recognition research [12]. This procedure involves the careful selection and design of very complex and time-consuming hand-crafted features for footstep recognition. The features include Geometric, Holistic, Spectral and Wavelet approaches to name a few [12]. Automatic feature learning models [13] have not been well studied for biometric footstep recognition using floor sensors systems.

Research studying footstep data as a biometric collected footstep signals from: (i) switch sensors [59, 60] which analyzes the spatial distribution of the footstep signals, and (ii) pressure sensors [26–28], focusing on dynamic pressure information in the signals, but with low spatial resolution. Qian et al. [61] use a commercial pressure mat with high resolution is used by in order to extract the center of pressure information, therefore using time and spatial pressure information only for some selected key points (geometric approach).

Recently, footstep signals in temporal and spatial domains were analyzed [29], reporting experiments on the SFootBD. The spatial information is extracted from accumulated pressure images. Temporal information was extracted from the average GRF and from other hand-crafted features. Principal Component Analysis (PCA) was used for dimensionality reduction of the footstep data and a non-linear SVM is used for biometric verification. Results were obtained in the range of 2.5–10% Equal Error Rate (EER) were achieved depending on the application setting. In [36] we reported a pilot study of a convolutional neural network model to learn processed spatial footstep features of the SFootBD database, suggesting significant improvements of footstep recognition performance compared to existing work [29].

Table 2 shows the recognition performance of the approach compared to other known biometric verification systems based on floor sensor data only. The other

**Fig. 8** Two-stream spatio-temporal resnet architecture for raw footstep representation



studies do not use the SFootBD database, thus cannot be directly compared to this work in terms of performance since the experiments differ in the number of clients and footstep signals. However, we are using a much larger database and therefore the performance results are more statistically significant.

In this report, we analyze the effect of evaluating a set of diverse footstep data representations in machine learning models. Two representations worked best overall for the spatio-temporal biometric verification problem presented here: raw footstep data and processed footstep data.

### 5.3 Deep Residual Network Model

The deep machine learning models used in this work are based on the state-of-the-art resnet architecture [62].

The resnet architecture is illustrated in Fig. 8 consisting of spatial and temporal streams for the raw representation. From input to output, each stream consists of the following layers: First, there is a resnet configuration 1 block (2ay) (Fig. 9 right), followed by resnet configuration 2 block (x2) (2by and 2cy) (Fig. 9 left), then an average pooling layer, fully connected layer (FC) and finally a softmax layer. The blocks consist of convolutional layers, batch normalization [63] and ReLU activation functions [64]. The residual units in the network can be expressed in general form as:

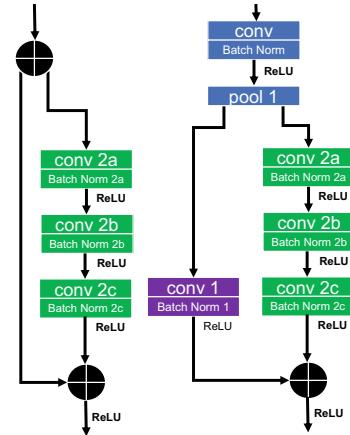
$$y_l = h(x_l) + G(x_l, W_l), \quad (2)$$

$$x_{l+1} = f(y_l), \quad (3)$$

where  $x_l$  is the input to the  $l$ -th residual block, and  $x_{l+1}$  is its corresponding output and  $G$  is a non-linear residual function.  $h(x_l) = x_l$  is an identity mapping,  $f$  is a RELU activation [64] function.  $W_l = \{W_{l,k} \mid 1 \leq k \leq K\}$  is the set of weights and biases of the  $l$ -th residual block.  $K$  is the number of layers in a residual unit. If  $f$  is an identity mapping, then  $x_{l+1} \equiv y_l$ , therefore Eq. 3 can be expressed as:

$$x_{l+1} = x_l + G(x_l, W_l). \quad (4)$$

**Fig. 9** Resnet model building blocks. Right: resnet configuration 1, Left: resnet configuration 2



For any unit of  $L$  and shallow unit  $l$ , the forward propagation of the feature  $x_L$  can be expressed as an additive output:

$$x_L = x_l + \sum_{i=l}^{L-1} G(x_i, W_i). \quad (5)$$

Therefore, during forward propagation,  $x_l$  is propagated to any  $x_L$  plus the residual factor. If the loss function is expressed as  $\gamma$ , the backpropagation of errors in the network can be expressed as the chain rule [65]:

$$\frac{\partial \gamma}{\partial x_l} = \frac{\partial \gamma}{\partial x_L} \frac{\partial x_L}{\partial x_l} = \frac{\partial \gamma}{\partial x_L} \left( 1 + \frac{\partial}{\partial x_L} \sum_{i=l}^{L-1} G(x_i, W_i) \right). \quad (6)$$

By using the Resnet as feature extractor, it eases the evaluation of the verification biometric system. This by evaluation the learn feature set with a discriminative linear classifier. This allowed saving computational resources and time. The linear classifier selected for the evaluation of the experiments was a linear Support Vector Machine (SVM), due to its high biometric performance when compared with other linear classifiers such as logistic regression or perception. If  $u$  is considered as the total number of clients for a given experiment, then is required to train  $u$  linear SVM classifier models using the Resnet models as a feature extractor, instead of training  $u$  Resnet models which are computationally expensive to train.

The RMSprop [66] optimizer was selected to update the model's weights due to its stability at training time. All models were trained with a Batch size of 32 samples. Initialization of the models with ImagNet Resnet-50 [67] weights for transfer learning was tested without major improvements, therefore the weights were initialized instead by sampling values from a Gaussian random distribution to ease the initialization process. The RMSprop learning rate was set initially at 0.001 and decreased by

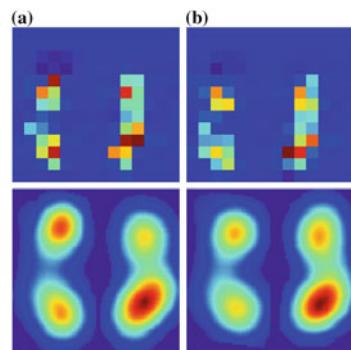
a factor of 10 once the learning error plateaus. An early learning stopping procedure was implemented: we stop training once the validation error stopped decreasing.

## 5.4 Spatial and Temporal Architectures

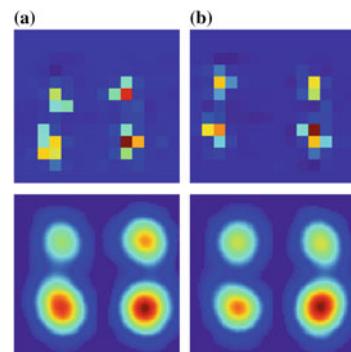
As footstep GRF patterns tend to contain a large degree of fine-grained GRF variability they are difficult to visualise for evaluation by humans Figs. 10 and 11 shows a side by side comparison of stride raw (top) and processed (bottom) spatial footstep representations from 2 clients of the SFootBD, considering 2 samples per user. The comparison implies that effective footstep recognition based only on visual perception is a very challenging problem as there can be a high user intra-variability and low inter-user variability in some cases. Moreover, humans are not accustomed to recognizing this type of images as opposed to other biometric traits such as facial recognition. Machine learning has been used in an attempt to solve differentiating the fine-grained GRF variability between clients and impostors.

The spatial and temporal footstep data share the same resnet architecture shown in Fig. 8. The input footstep representations affect the dimensions of the first

**Fig. 10** Spatial raw (top) and spatial processed (bottom) footstep representations of user 1. **a** Sample 1 **b** sample 2. Top representation dimension is  $13 \times 14$  pixels. Bottom representation dimension is  $88 \times 88$  pixels



**Fig. 11** Spatial raw (top) and spatial processed (bottom) footstep representations of user 2. **a** Sample 1 **b** sample 2. Top representation dimension is  $13 \times 14$  pixels. Bottom representation dimension is  $88 \times 88$  pixels



convolutional (conv.) layer of the resnet model, it takes as input a stride footstep tensor of shape  $(n, m, c)$  where  $n \times m$  is the 2D footstep sensor matrix and  $c$  the frames.  $c = 1$  for the spatial case and  $c = 100$  for the temporal component. The filter size of the resnet blocks (Fig. 9) and channels change according to the input footstep tensor dimensions.

The widely-used deep network design introduced by the VGG net [68] is adopted for the resnet models. The methodology decreases the spatial component at the conv. layers as a function of increasing the number of filter maps, from the left (input) to the right (output) layers of the network.

## 5.5 Verification System Evaluation

The verification system performance was evaluated by using the Detection error trade-off (DET) curve [69], which displays a trade-off of missed detection and false alarm errors. We also used the Equal Error Rate (EER) to summarise the biometric verification performance of the system. The EER is the intersection in the DET curve where the False Rejection Rate (FRR) and the False Acceptance Rate (FAR) are equal. Therefore, we are giving equal importance to FRR and FAR for the evaluation of our experiments.

## 5.6 Results

### 5.6.1 Airport Scenario: Benchmark B1

For this benchmark, the fusion of the spatial and temporal domains performs best overall for the 3 representations considered. Separately, the raw representation delivered the best performance (11.80%, 11.50%) EER followed by the processed SVM representation with (8%, 12.50%) EER and lastly, the processed representation obtained (10.10%, 14.50%) EER.

For the fusion of representations, the raw and processed representations deliver (8.10%, 10.70%) EER. This is also a better performance than considering the two representations separately. While the combination of the raw, processed and processed SVM representations delivers the optimal performance overall (7.10%, 10.50%) EER. This improves the previous reported optimal performance [29] by 2% EER in evaluation and 0.9% EER in validation datasets. This benchmark considers the least amount of footstep data for training from the 3 benchmarks. The benchmark exemplifies a real-world security application, where data is scarce.

### 5.6.2 Workplace Scenario: Benchmark B2

Spatio-temporal fusion performs best overall for the 3 representations in this dataset. The processed SVM delivers the best performance from the 3 representations with (3.80%, 6.70%) EER. The raw and processed SVM representation delivers the same evaluation performance of 8% EER. In validation, the raw representation obtains 6.10% EER while the processed SVM delivers better performance of 3.80% EER.

For the fusion of representations, the raw and processed representations deliver (3.20%, 5.30%) EER this performs better than any of the representations considered. The combination of the raw, processed and processed SVM representations deliver the optimal performance overall of (2.80%, 4.90%) EER for this dataset. This improves the previous reported optimal performance [29] by 1.8% EER in evaluation and 1% EER in validation datasets

This benchmark considers a medium amount of footstep data for training from the 3 benchmarks. An office security environment exemplifies a real-world scenario.

### 5.7 Home Scenario: Benchmark B3

Spatio-temporal fusion performs best overall for the 3 representations. The processed representation delivers the best performance with (1.80%, 2.60%) EER. The processed SVM follows with (2.10%, 3.20%) EER and lastly, the raw representation obtained (1.70%, 5.60%) EER.

At the fusion of representations level, the raw and processed representations deliver (0.80%, 2.10%) EER performing better than considering the representations separately as in previous benchmarks. The combination of the raw, processed and processed SVM representations deliver the optimal performance overall (0.70%, 1.70%) EER for this dataset and overall in all experiments. This improves the previous reported optimal performance [29] by 2.3% in the evaluation and 1.4% in validation datasets These results are the best overall considering all experiments and benchmarks.

This benchmark considers the largest amount of footstep data for training from the 3 benchmarks, thus the best performance observed overall experiments.

We argue that the best performance observed here overall experiments is since the largest amount of footstep data is considered for training the Resnet models. A home environment exemplifies a real-world security application of this dataset, and where the proposed methodology and models would optimally work.

### 5.8 Discussion

The partition of the test datasets into validation and evaluation subsets allows evaluation of the model's generalisation performance with high confidence since the

**Table 9** Biometric verification results in terms of EER (in %) for benchmarks B1, B2 and B3

Domain	Model	Benchmark B1 (40 clients)		Benchmark B2 (15 clients)		Benchmark B3 (5 clients)	
		Val. (%)	Eval. (%)	Val. (%)	Eval. (%)	Val. (%)	Eval. (%)
<i>Raw representations</i>							
Temporal	Resnet	14.70	18.00	8.20	6.70	4.60	8.00
Spatial	Resnet	16.30	13.40	11.20	10.70	3.40	12.00
Spatio-temporal	Resnet	11.80	11.50	6.10	8.00	1.70	5.60
Spatio-temporal	DNN	27.65	27.93	14.33	17.33	5.76	6.57
Spatio-temporal	CNN	31.28	31.21	14.26	14.67	3.62	4
<i>Processed representations</i>							
Temporal	Resnet	12.20	18.00	6.60	9.30	3.90	2.00
Spatial	Resnet	13.60	15.50	5.50	9.30	3.00	6.60
Spatio-temporal	Resnet	10.10	14.50	3.80	8.00	1.80	2.60
Spatio-temporal	DNN	17.25	21	6.10	6.66	2.80	3
Spatio-temporal	CNN	18.1	23	6.07	9.95	1.61	3.38
<i>Processed SVM representations</i>							
Spatial-integrated temporal	SVM	12.10	16.50	9.30	12.00	6.10	8.20
Spatial	SVM	11.70	17.50	5.90	9.20	3.80	2.60
Spatio-temporal	SVM	8.00	12.50	3.80	6.70	2.10	3.20
<i>Fusion of representations</i>							
Raw and processed	Resnet	8.10	10.70	3.20	5.30	0.80	2.10
Raw and processed and processed SVM	Resnet and SVM	7.10	10.50	2.80	4.90	0.70	1.70

evaluation dataset never influence the training process directly (training set) or indirectly (validation set). Overall, the validation dataset EER is better than the evaluation dataset due to the generalisation of the model in held-out footstep data. We are able to provide better performance results in all benchmarks when compared with previously reported work [29].

The validation dataset performance influences the early stopping procedure at the training time of the resnet models, thus indirectly influencing the generalization performance of the system. However, this is a widely used procedure, and by providing an EER performance in a held-out dataset (evaluation) a closer and more realistic estimate of the generalization performance is provided.

Deep residual networks are known to show state-of-the-art performance for problems that use large amounts of footstep data for model training, such as ImageNet [13, 67] which contains millions of samples for training. This effect can be shown for both the validation and evaluation dataset performance results shown in Table 9, as data available per model increases.

The raw and processed resnet representations obtained very similar performance EER in the 3 datasets as observed in Table 9. Therefore, the raw models are able to provide competitive performance from raw unprocessed footstep data evaluated in a learning model when compared with processed footstep data.

This section has explored the important effects of testing spatio-temporal input footstep data representations in machine learning models based on deep residual networks. The representations are based on footstep raw and processed data. We compare its performance with a processed representations approach using a SVM. The two methods delivered similar performance. The critical factors that affect footstep biometric verification performance are the spatio-temporal data representations considered and the amount of data considered for training.

Three datasets from the largest footstep database were considered for the spatio-temporal analysis. The dataset resembles data-driven real-world scenarios, including a small footstep dataset for security applications (Benchmark B1), a medium size dataset for office-oriented applications (Benchmark B2), and a large dataset for home-based scenarios (Benchmark B3). These scenarios intend to cover the most common real-world scenarios.

The experiments performed here have proven that there is not a single optimal representation for all datasets. Considering the representations separately, for Benchmark B1 the raw representation performs optimally, in Benchmark B2 the processed SVM delivers optimal verification performance and for Benchmark B3 the processed SVM representation performs best overall, this justifies this research in terms of evaluation of several representations in machine learning models in order to obtain a robust footstep recognition model.

This result highlights the need for raw data representation analysis for automatic feature learning models. We have demonstrated that an ensemble of resnet and SVM models using processed and unprocessed footstep data obtain a robust footstep recognition model for biometric verification.

## 6 Conclusions

In this chapter spatio-temporal gait and footstep representations have been studied with deep learning methodologies. In the healthcare theme, dual-task has been classified with robust classification performance by providing an F-score of 97.33% in the optimal case, while in the security theme, state-of-the-art footstep recognition performance has been obtained in a biometric verification scenario, obtaining an optimal EER of 0.7%. Therefore, robust pattern recognition in gait and footstep analysis have been provided with high statistical significance. The methodologies to obtain the optimal results used deep machine learning principles based on convolutional neural networks.

In the healthcare theme, the link between cognitive activities and their effects on the changes in human gait patterns was investigated. The research analyzed of the effect of cognitive activities in gait patterns from healthy individuals. The

methodology delivered results with a cohort of 69 participants performing dual-tasks experiments. In the optimal case scenario, an F-score of 97% was obtained to identify dual-tasks patterns. The methodology clearly outperformed optimized classical machine learning models (non-deep learning) and was able to distinguish the gender of participants with an optimal F-score of 97.3%.

In the security theme, state-of-the-art footstep recognition performance results were obtained in challenging biometric verification scenarios. The largest to date footstep database, the sfootBD, was used to validate the deep machine learning methodology. The database has almost 20,000 footstep signals from 127 users. First, the spatial footstep domain was studied in a single real-world biometric verification setting, then the spatio-temporal footstep domain was studied extensively in three critical real-world biometric verification scenarios: at home, office and airport scenarios. The optimal results demonstrated that an ensemble of processed and raw footstep data and a combination of shallow and deep machine learning models delivered state-of-the-art recognition performance for biometric verification. The methodology delivered a 0.7% EER in the optimal biometric verification case.

The methodology presented in the healthcare theme may be potentially applied to studies of large cohorts of users in the MCI stage or with the pathology of AD [57, 70]. This research direction could further investigate the link between changes in gait and neurodegenerative disease progression at early stages. The deep learning methodologies presented here can be applied in a multi-sensor spatio-temporal environment to search for further behavioral signatures that may flag AD in the prodromal stage. This approach could include other sensing modalities for example by integrating keyboard use patterns of individuals in daily computer use [71] or speech recognition analysis [72]. Furthermore, accelerometer sensors, cameras or indoor/outdoor tracking systems [70] could also be integrated for robust analysis of early stages of AD.

**Acknowledgements** We express our gratitude to the participants for taking the time to participate in this research and to David H. Foster for useful discussions. This work was supported by the U.K. Engineering and Physical Sciences Research Council EP/K005294/1 EP/K503447/1, in part by CONACyT (Mexico), grant 467373 and in part by the University of Manchester Data Science Institute. O. Costilla-Reyes would like to acknowledge CONACyT (Mexico) for a studentship. We acknowledge NVIDIA for the donation of the GPU used to perform some of the experiments of this research.

## References

1. Michael, W., Whittle. *Gait Analysis: An Introduction*. Butterworth, Heinemann, (2014)
2. Nadkarni, N.K., Mawji, E., McIlroy W.E, Black S.E.: Spatial and temporal gait parameters in Alzheimer's disease and aging. *Gait and Posture* **30**(4), 452–454 (2009)
3. Hollman, J.H., Kovash, F.M., Kubik, J.J., Linbo, R.A.: Age-related differences in spatiotemporal markers of gait stability during dual task walking. *Gait and Posture* **26**(1), 113–119 (2007)

4. Beurskens, R., Bock, O.: Age-related deficits of dual-task walking: a review. *Neural Plast.* **2012** (2012)
5. Hausdorff, J.M., Schweiger, A., Herman, T., Yogev-Seligmann, G., Giladi, N.: Dual-task decrements in gait: contributing factors among healthy older adults. *J. Gerontol. Ser. A: Biol. Sci. Med. Sci.* **63**(12), 1335–1343 (2008)
6. Barnes, D.E., Yaffe, K.: The projected effect of risk factor reduction on Alzheimer's disease prevalence. *Lancet Neurol.* **10**(9), 819–828 (2011)
7. Lundin-Olsson, L., Nyberg, L., Gustafson, Y.: Stops walking when talking as a predictor of falls in elderly people. *Lancet* **349**(9052), 617 (1997)
8. Costilla-Reyes, O., Vera-Rodriguez, R., Scully, P., Ozanyan, K.B.: Analysis of Spatio-temporal representations for robust footprint recognition with deep residual neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **41**(2), 285–296 (2018)
9. Vacca, J.R.: *Biometric Technologies and Verification Systems*. Butterworth, Heinemann (2007)
10. P. Daphne Tsatsoulis, Jaech, A., Batie, R., Savvides, M.. *Continuous authentication using biometrics*. IGI Global, 68–88 (2012)
11. Muro-de-la Herran, A., Garcia-Zapirain, B., Mendez-Zorrilla, A.: Gait analysis methods: an overview of wearable and non-wearable systems, highlighting clinical applications. *Sensors* **14**(2), 3362–3394 (2014)
12. Eric Mason, J., Traoré, I., Woungang, I.: *Machine Learning Techniques for Gait Biometric Recognition*. Springer (2016)
13. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
14. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al.: Imagenet large scale visual recognition challenge. *arXiv preprint arXiv:1409.0575* (2014)
15. J.M. Montepare, Goldstein, S.B.: The identification of emotions from gait information. *J. Nonverbal Behav.* **11**(1), 33–42 (2016)
16. Martina, Z., Simon,H., Wilkowska, W.: *When Your Living Space Knows What You Do: Acceptance of Medical Home Monitoring by Different Technologies*. Springer (2011)
17. Alharthi, A.S., Yunas, S.U., Ozanyan, K.B.: Deep learning for monitoring of human gait: a review. *IEEE Sens. J.* (submitted) (2019)
18. Connor, P., Ross, A.: Biometric recognition by gait: a survey of modalities and features. *Comput. Vis. Image Underst.* **167**, 1–27 (2018)
19. El-Alfy, H., Mitsugami, I., Yagi, Y.: A new gait-based identification method using local Gauss maps. In: *Asian Conference on Computer Vision*, pp. 3–18. Springer (2014)
20. Ioannidis, D., Tzovaras, D., Damousis, I.G., Argyropoulos, S., Moustakas, K.: Gait recognition using compact feature extraction transforms and depth information. *IEEE Trans. Inf. Forensics Secur.* **2**, 623–630 (2007)
21. Arora, P., Srivastava, S., Arora, K., Bareja, S.: Improved gait recognition using gradient histogram Gaussian image. *Procedia Comput. Sci.* **58**, 408–413 (2015)
22. Liu, Y., Zhang, J., Wang, C., Wang, L.: Multiple HOG templates for gait recognition. *2012 21st International Conference on Pattern Recognition (ICPR)*, pp. 2930–2933. IEEE (2012)
23. Sivapalan, S., Chen, D., Denman, S., Sridharan, S., Fookes, C.: Gait energy volumes and frontal gait recognition using depth images. In: *2011 International Joint Conference on Biometrics (IJCB)*, pp. 1–6. IEEE (2011)
24. Costilla-Reyes, O., Scully, P., Ozanyan, K.B.: Temporal pattern recognition in gait activities recorded with a footprint imaging sensor system. *IEEE Sens. J.* **16**(24), 8815–8822 (2016)
25. Costilla-Reyes, O., Scully, P., Ozanyan, K.B.: Deep neural networks for learning spatio-temporal features from tomography sensors. *IEEE Trans. Ind. Electron.* **65**(1), 645–653 (2018)
26. Cattin, P.C.: Biometric authentication system using human gait. PhD thesis. Diss., ETH Zurich, Nr. 14603, pp. 1–140 (2002)
27. Stevenson, J.P., Firebaugh, S.L., Charles, H.K.. Biometric identification from a floor based PVDF sensor array using hidden Markov models. *Proc. SAS* **7** (2007)
28. Mostayed, A., Kim, S., Mazumder, M.M.G., Park, S.J.: Foot step based person identification using histogram similarity and wavelet decomposition. In: *Proceedings of the 2nd International Conference on Information Security and Assurance*, , pp. 307–311. IEEE (2008)

29. Vera-Rodriguez, R., Mason, J.S.D., Fierrez, J., Ortega-Garcia, J.: Comparative analysis and fusion of spatiotemporal information for footstep recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(4), 823–834 (2013)
30. Zhong, Y., Deng, Y.: Sensor orientation invariant mobile gait biometrics. In: *2014 IEEE International Joint Conference on Biometrics (IJCB)*, pp. 1–8. IEEE (2014)
31. Zhang, Y., Pan, G., Jia, K., Lu, M., Wang, Y., Wu, Z.: Accelerometer based gait recognition by sparse representation of signature points with clusters'. *IEEE Trans. Cybern.* **45**(9), 1864–1875 (2015)
32. Bours, P., Shrestha, R.: Eigensteps: a giant leap for gait recognition. In: *2010 2nd International Workshop on Security and Communication Networks (IWSCN)*, pp. 1–6. IEEE (2010)
33. Gafurov, D., Snekkenes, E., Bours, P.: Improved gait recognition performance using cycle matching. In: *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 836–841. IEEE (2010)
34. Rong, L., Jianzhong, Z., Ming, L., Xiangfeng, H.: A wearable acceleration sensor system for gait recognition. In: *2007 2nd IEEE Conference on Industrial Electronics and Applications, ICIEA 2007*, pp. 2654–2659. IEEE (2007)
35. Zifeng, W., Huang, Y., Wang, L., Wang, X., Tan, T.: A comprehensive study on cross-view gait based human identification with deep CNNs. *IEEE Trans. Pattern Anal. Mach. Intell.* **39**(2), 209–226 (2017)
36. Costilla-Reyes, O., Vera-Rodriguez, R., Scully, P., Ozanyan, K.B.: Spatial footstep recognition by convolutional neural networks for biometric applications. In: *Proceedings of IEEE SENSORS 2016*. IEEE (2016)
37. Yun, J.: User identification using gait patterns on UbiFloorII. *Sensors* **11**(3), 2611–2639 (2011)
38. Claude Cattin, P.: Biometric authentication system using human gait. PhD thesis. PhD dissertation Technische Wissenschaften ETH Zurich Nr. 14603 (2002)
39. Kanth Saripalle, S.: Classification of human postural and gestural movements using center of pressure parameters derived from force platforms. PhD thesis. University of Missouri- Kansas City (2010)
40. Headon, R., Curwen, R.: Recognizing movements from the ground reaction force. In: *Proceedings of the 2001 Workshop on Perceptive User Interfaces*, pp. 1–8. ACM (2001)
41. Simeone, O.: A very brief introduction to machine learning with applications to communication systems. In: *IEEE Transactions on Cognitive Communications and Networking* (2018)
42. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web (1999)
43. Zhao, W., Krishnaswamy, A., Chellappa, R., Swets, D.L., Weng, J.: Discriminant analysis of principal components for face recognition. In: *Face Recognition*, pp. 73–85. Springer (1998)
44. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **17**(6), 734–749 (2005)
45. Yi, N., Li, C., Feng, X., Shi, M.: Research and improvement of convolutional neural network. In: *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pp. 637–640. IEEE (2018)
46. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. *Adv. Neural Inf. Process. Syst.*, 568–576 (2014)
47. Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., Darrell, T.: Long-term recurrent convolutional networks for visual recognition and description. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2625–2634 (2015)
48. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3d convolutional networks. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4489–4497 (2015)
49. Feichtenhofer, C., Pinz, A., Wildes, R.: Spatiotemporal residual networks for video action recognition. *Adv. Neural Inf. Process. Syst.*, 3468–3476 (2016)

50. Ji, S., Wei, X., Yang, M., Kai, Y.: 3D convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(1), 221–231 (2013)
51. Wang, X., Farhadi, A., Gupta, A.: Actions—transformations. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2658–2667 (2016)
52. Gafurov, D.: A survey of biometric gait recognition: approaches, security and challenges. In: *Annual Norwegian Computer Science Conference*, pp. 19–21. Citeseer (2007)
53. Montero-Odasso, M., Muir, S.W., Speechley, M.: Dual-task complexity affects gait in people with mild cognitive impairment: the interplay between gait variability, dual tasking, and risk of falls. *Arch. Phys. Med. Rehabil.* **93**(2), 293–299 (2012)
54. Owings, T.M., Grabiner, M.D.: Measuring step kinematic variability on an instrumented treadmill: how many steps are enough? *J. Biomech.* **36**(8), 1215–1218 (2003)
55. Atkinson, H.H., Rosano, C., Simonsick, E.M., Williamson, J.D., Davis, C., Ambrosius, W.T., S.R. Rapp, Cesari, M., Newman, A.B., Harris, T.B.: Cognitive function, gait speed decline, and comorbidities: the health, aging and body composition study. *J. Gerontol. Ser. A: Biol. Sci. Med. Sci.* **62**(8), 844–850 (2007)
56. Powers, D.M.W.: Evaluation: from precision, recall and F-measure to Roc, informedness, markedness & correlation. *J. Mach. Learn. Technol.* **2**(1), 37–63 (2011)
57. Gunn-Moore, D., Kaidanovich-Beilin, O., Iradi, M.C.G., Gunn-Moore, F., Lovestone, S.: Alzheimer’s disease in humans and other animals: a conseqREFERENCES 33 quence of postreproductive life span and longevity rather than aging. *Alzheimer’s Dement.: J. Alzheimer’s Assoc.* **14**(2), 195–204 (2018)
58. Allan, L.M., Ballard, C.G., Burn, D.J., Anne Kenny, R.: Prevalence and severity of gait disorders in Alzheimer’s and non-Alzheimer’s dementias. *J. Am. Geriatr. Soc.* **53**(10), 1681–1687 (2005)
59. Middleton, L., Buss, A., Bazin, A., Nixon, M.: A floor sensor system for gait recognition. In: *Proceedings of Fourth IEEE Workshop on Automatic Identification Advanced Technologies*, pp. 171–176 (2005)
60. Vera-Rodriguez, R., Fierrez, J., Mason, J.S.D., Ortega-Garcia, J.: A novel approach of gait recognition through fusion with footstep information. In: *Proceedings IAPR International Conference on Biometrics*, ICB (2013)
61. Qian, G., Zhang, J., Kidane, A.: People identification using floor pressure sensing and analysis. *IEEE Sens. J.* **10**(9), 1447–1460 (2010)
62. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778 (2016)
63. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456 (2015)
64. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 315–323 (2011)
65. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: *Proceedings of European Conference on Computer Vision*, pp. 630–645. Springer (2016)
66. Dauphin, Y., de Vries, H., Bengio, Y.: Equilibrated adaptive learning rates for non-convex optimization. *Adv. Neural Inf. Process. Syst.* , 1504–1512 (2015)
67. Russakovsky, O., Deng, J., Hao, S.: Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* **115**(3), 211–252 (2015)
68. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: *International Conference on Learning Representations (ICRL)*, pp. 1–14 (2015)
69. Martin, A., Doddington, G., Kamm, T., Ordowski, M., Przybocki, M.: The DET curve in assessment of detection task performance. *Tech. Rep.*, DTIC (1997)
70. Lawson, J., Murray, M., Zamboni, G., Koychev, I.G., Ritchie, C.W., Ridha, B.H., Rowe, J.B., Thomas, A., Ffythe, D.H., Howard, R.J.: Deep and frequent phenotyping: a feasibility study for experimental medicine in dementia. *Alzheimer’s Dement.: J. Alzheimer’s Assoc.* **13**(7), pp. 1268–1269 (2017)

71. Gledson, A., Asfiandy, D., Mellor, J., Ba-Dhfari, T.O.F., Stringer, G., Couth, S., Burns, A., Leroi, I., Zeng, X., Keane, J.: Combining mouse and keyboard events with higher level desktop actions to detect mild cognitive impairment. In: *2016 IEEE International Conference on Healthcare Informatics (ICHI)*, pp. 139–145. IEEE (2016)
72. Aharon Satt, A., Sorin, A., Hoory, R., Toledo-Ronen, O., Derreumaux, A., Manera, V., Verhey, F., Aalten, P., Robert, P.H.: Automatic speech analysis for the assessment of patients with predementia and Alzheimer's disease. *Alzheimer's Dement.: Diagn., Assess. Dis. Monit.* **1**(1), 112–124 (2015)

# Deep Learning for Building Occupancy Estimation Using Environmental Sensors



Zhenghua Chen, Chaoyang Jiang, Mustafa K. Masood, Yeng Chai Soh,  
Min Wu and Xiaoli Li

**Abstract** Building Energy efficiency has gained more and more attention in last few years. Occupancy level is a key factor for achieving building energy efficiency, which directly affects energy-related control systems in buildings. Among varieties of sensors for occupancy estimation, environmental sensors have unique properties of non-intrusion and low-cost. In general, occupancy estimation using environmental sensors contains feature engineering and learning. The traditional feature extraction requires to manually extract significant features without any guidelines. This handcrafted feature extraction process requires strong domain knowledge and will inevitably miss useful and implicit features. To solve these problems, this chapter presents a Convolutional Deep Bi-directional Long Short-Term Memory (CDBLSTM) method that consists of a convolutional neural network with stacked architecture to automatically learn local sequential features from raw environmental sensor data from scratch. Then, the LSTM network is used to encode temporal dependencies of these local features, and the Bi-directional structure is employed to consider the past and future contexts simultaneously during feature learning. We conduct real experiments

---

Z. Chen (✉) · M. Wu · X. Li

Institute for Infocomm Research (I2R), A\*STAR, Singapore, Singapore

e-mail: [Chen\\_Zhenghua@i2r.a-star.edu.sg](mailto:Chen_Zhenghua@i2r.a-star.edu.sg)

M. Wu

e-mail: [wumin@i2r.a-star.edu.sg](mailto:wumin@i2r.a-star.edu.sg)

X. Li

e-mail: [xlli@i2r.a-star.edu.sg](mailto:xlli@i2r.a-star.edu.sg)

C. Jiang

School of Mechanical Engineering, Beijing Institute of Technology, Beijing, China

e-mail: [cjiang@bit.edu.cn](mailto:cjiang@bit.edu.cn)

M. K. Masood

Department of Civil Engineering, Aalto University, Espoo, Finland

e-mail: [nkm\\_imn@hotmail.com](mailto:nkm_imn@hotmail.com)

Y. C. Soh

School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, Singapore

e-mail: [eycsoh@ntu.edu.sg](mailto:eycsoh@ntu.edu.sg)

to compare the CDBLSTM and some state-of-the-art approaches for building occupancy estimation. The results indicate that the CDBLSTM approach outperforms all the state-of-the-arts.

**Keywords** Deep learning · Building occupancy estimation · Environmental sensors · CDBLSTM

## 1 Introduction

To maintain the thermal comfort of indoor environments, around 40% of the energy has been consumed in building sectors [28]. Thus, a lot of attention has been paid on building energy efficiency and sustainable development. To achieve that, a crucial factor is the building occupancy information, also known as occupant number or range in buildings. It can be used for building climate and adaptive light control [28, 36]. Balaji et al. saved 17.8% of energy for HVAC systems relied on actual occupancy levels in a designed experiment [1]. A light control system developed in [24] has reported a reduction of 35–75% of energy consumption for building light control systems. However, to obtain an accurate and robust occupancy estimation system is a challenging mission and remain unsolved.

Occupancy estimation can be done by the use of different sensors. For instance, Liu et al. present a detection of the absence and presence of occupants via PIR sensors [27]. It will be more meaningful to obtain the actual occupant number or range indoors. In order to fulfill that, the methods relied on RFID and wearable devices were presented in [1, 25]. However, these approaches require users to wear specific devices, which is intrusive and inconvenient. Accurate occupancy estimation can be achieved by using cameras [42]. However, camera based solutions often suffer from the problems of insufficient illumination and high computational load. Besides, they also have the issue of privacy concerns. Some other methodologies rely on occupants' involvement, such as using chair sensors [23] and applicants power usage data [22]. However, occupants that do not involved will not be able to be detected.

Recently, environmental sensors are widely adopted for occupancy estimation, because they are low-cost and non-intrusive for users [21, 29, 40, 41]. Due to the complex relationship between environmental sensor measurements and occupancy levels, physical modeling is with limited performance. An alternative way is to model the complex relationship by using machine learning techniques which work well on function approximation. Since, environmental sensor data are with large noise and not representative for different occupancy levels, the machine learning models trained with raw sensory data may have limited performance. The common operation is to perform feature engineering which intends to extract more informative representations for different occupancy levels [26]. However, the traditional manual feature engineering does not have a guideline on which features should be extracted for occupancy inference. In addition, it requires strong domain knowledge and will inevitably miss implicit and useful features. To solve this problem, this

chapter presents a Convolutional Deep Bi-directional Long Short-Term Memory (CDBLSTM) that consists of a convolutional neural network with a stacked structure to learn useful representations (features) automatically from scratch [11]. The convolutional network is able to learn some sequential local features from raw environmental sensor data. Since the environmental sensory data is a typical time series, temporal dependencies are of great importance for accurate and robust occupancy inference. To model the temporal dependencies in data, we adopt a BLSTM network whose inputs are the sequential local features learned by the convolutional neural network. We have compared the CDBLSTM approach with some state-of-the-arts in existing literature by using real evaluation.

## 2 Literature Review

Many advanced algorithms have been presented for occupancy inferences in buildings using environmental sensor data. The authors in [13] presented an occupancy estimation system for an open office room by using sensor networks that are able to collect data of CO<sub>2</sub>, CO, acoustics, PM2.5, motion, illumination, temperature and humidity. Some statistical features, e.g., moving average of 20-min and 1st order difference, were manually extracted. Next, the most important features were chosen via the popular information gain theory. Finally, data-driven methods including Support Vector Machine (SVM), Artificial Neural Network (ANN) and Hidden Markov Model (HMM) were utilized for occupancy estimation. They made a conclusion that the most significant sensors are CO<sub>2</sub> and acoustic, and the HMM achieves the best performance for occupancy estimation.

The authors in [30] employed environmental sensors of temperature, CO<sub>2</sub>, humidity, and pressure, to estimate occupancy for a tutorial room. They extracted some similar features used in [13]. An ELM-based wrapper algorithm was developed for feature selection and occupancy inference.

In [38], the authors investigated various sensors including sound, motion, temperature, door state, CO<sub>2</sub>, humidity, passive infrared and light to infer occupancy in both multi-occupant and single-occupant offices via some widely used machine learning algorithms. Instead of extracting more useful features, they used raw sensor data as features. Here, the authors applied many informative sensors to guarantee a satisfactory performance of their proposed method. The contribution of different sensors (features) were tested by using the theory of information gain. Eventually, light level, door state and CO<sub>2</sub> are shown to be the most important parameters. For different algorithms, the decision tree (DT) approach has the best performance.

Candanedo et al. developed an occupancy detection system with sensors of humidity, CO<sub>2</sub>, temperature and light levels [3]. They also used the raw sensor data as features in this work, and utilized some statistical models identify the two states of absence and presence of occupants. Different combinations of features with distinct statistical approaches were tried, and then the best sensors and models can be selected. At last, they made a conclusion which claims that a satisfactory performance is able to be fulfilled when properly selecting sensors and learning methods.

Since occupancy dynamics has the Markov property [4, 7, 8], the HMM model has achieved great success for building occupancy detection and estimation [13]. But, the traditional HMM often suffers from some limitations, such as the use of mixture of Gaussian model to estimate emission probabilities and the fixed transition probability matrix. To solve these issues, the authors in [12] presented an IHMM-MLR for environmental sensor based occupancy inference. Firstly, inhomogeneous transition probability matrices for capturing occupancy dynamics at distinct time steps were developed. Then, multinomial logistic regression to produce the emission probabilities with environmental sensor data was designed. Two schemes, i.e., online and offline, were formulated to infer occupancy in distinct situations.

Chen et al. presented another system to enhance the performance for occupancy estimation by considering occupancy properties [6]. They performed a fusion of traditional machine learning algorithms with a well-developed occupancy model which is able to show occupancy properties. The sensors they utilized include CO<sub>2</sub>, humidity, pressure and temperature, which is widely available. The algorithms include ELM, SVM, ANN, KNN, CART and LDA. They formulated a Bayes filter to fuse the occupancy model and six data-driven algorithms for the estimation of occupancy. A detailed survey for occupancy estimation can be found in [5].

Here, we leverage on the environmental sensors including temperature, CO<sub>2</sub>, pressure and humidity that are popular in normal HVAC systems [14] instead of applying specific sensors, such as acoustic level [13, 38], motion [19, 38] and light level [3]. Without applying the noisy sensor data as features or using some handcrafted statistical features, we attempt to automatically extract some useful local sequential features by using the convolutional neural network with stacked structure. Then, the BLSTM network is able to encode temporal dependencies for sequential local features during high-level feature learning. We have made a comprehensive comparison with some state-of-the-arts by using actual experiments.

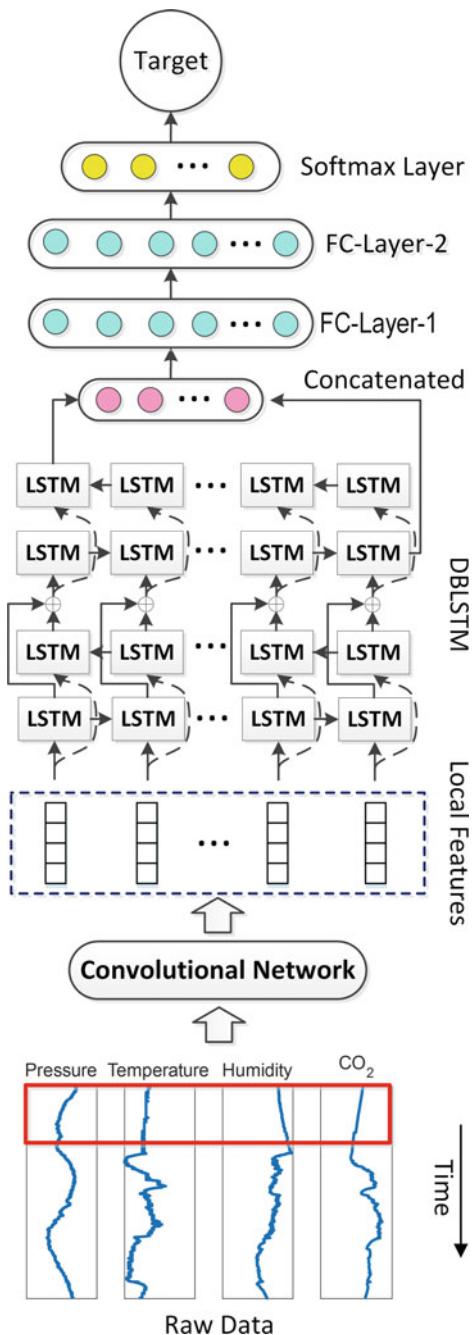
### 3 Methodology

We firstly demonstrate an overview of the CDBLSTM for environmental sensor based occupancy inference. Then, we introduce the key components in CDBLSTM, i.e., the convolutional neural network, the DBLSTM, and the classification layers. Finally, the introduction of the training process of the CDBLSTM approach will be covered.

#### 3.1 Overview

For environmental sensor based occupancy estimation, the key part is to learn discriminative representations (features) from raw data for distinct occupancy levels. Figure 1 presents the CDBLSTM framework for environmental sensor based occupancy

**Fig. 1** Framework of the CDBLSTM approach [11]



inference. Raw input is a sliding window of environmental sensor data. Then, a convolutional network with multiple filters is applied for learning features of local sliding windows known as local feature learning, which is of great importance for distinguishing data from different occupancy levels. Next, the DBLSTM is leveraged to encode temporal dependencies of local sequential features in forward and backward directions. Finally, the learned high-level features from the DBLSTM are fed into fully connected and softmax layers for the classification of different occupancy levels.

### 3.2 Convolutional Operation

We implement convolutional neural network on environmental sensor data to produce sequential local features. Generally, it contains a convolutional layer, together with a pooling layer. Figure 2 shows the convolutional and pooling operations on environmental sensor data. The functionality of the convolutional operation is to use a sliding window over the raw time-series data to get sequential local features. And then, the pooling operation is to reduce feature dimension of the sequential local features. The detailed implementation of the two operations will be presented below.

**Convolutional Layer:** Suppose that the  $n$  input samples are  $\{\mathbf{X}_i\}$ ,  $i = 1, 2, \dots, n$ , and each input sample  $\mathbf{X}_i \in \mathbb{R}^{r \times d}$  is a sliding window environment sensor data, where  $r$  is the length of sequence and  $d$  is the number of sensors. It can also be represented as  $\mathbf{X}_i = [\mathbf{x}_1, \dots, \mathbf{x}_r]$ . The definition of the convolution operation is to multiply a filter vector  $\mathbf{v} \in \mathbb{R}^{md \times 1}$  with a slice of the input  $\mathbf{x}_{i:i+m-1} \in \mathbb{R}^{md \times 1}$  which is shown as follows

$$\mathbf{x}_{i:i+m-1} = \mathbf{x}_i \oplus \mathbf{x}_{i+1} \oplus \dots \oplus \mathbf{x}_{i+m-1} \quad (1)$$

where  $m$  denotes the windows size and  $\oplus$  is the concatenation operation. Next, an activation function is performed over the multiplied results, shown as

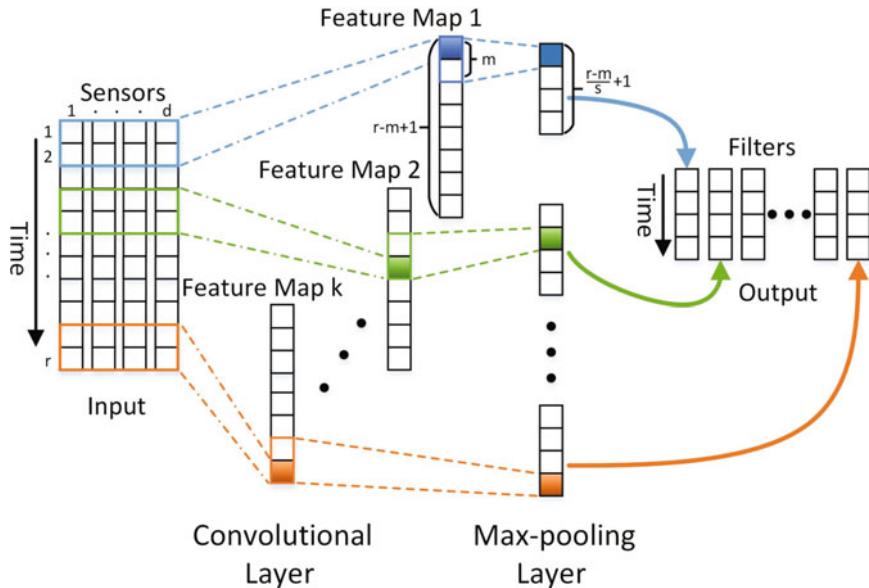
$$c_i = g(\mathbf{v}^\top \mathbf{x}_{i:i+m-1} + b) \quad (2)$$

where  $g(\cdot)$  is the activation function,  $b$  is the bias term and  $\top$  is the transpose operation. The widely used ReLU activation function [31] is adopted. By sliding the filter from the beginning of the input sequence to its end, we can produce a feature map, shown as follows:

$$\mathbf{c}^j = [c_1, c_2, \dots, c_{r-m+1}] \quad (3)$$

where  $j = 1, 2, \dots, k$ , and  $k$  is the number of filters.

**Pooling Layer:** The pooling operation is to reduce feature dimension, leading to more discriminative features [15]. In this work, we adopt the widely used max-pooling



**Fig. 2** Convolutional network structure

which conducts an operation of maximum on  $s$  consecutive components of feature map  $\mathbf{c}^j$ . After pooling operation, the features will be

$$\mathbf{z}^j = [z_1, z_2, \dots, z_{\frac{r-m}{s}+1}] \quad (4)$$

where  $z_i = \max(c_{is-s}, c_{is-s+1}, \dots, c_{is-1})$ . Hence, the pooling operation will generate compressed feature map  $\mathbf{z}^j$ ,  $j \in 1, 2, \dots, k$ . Eventually, the output of the convolutional neural network will have a feature dimension of  $(\frac{r-m}{s} + 1) \times k$ .

In general, assume the number of samples  $n$ , the input data has a dimension of  $n \times r \times d$ . The output of the convolutional neural network has a size of  $n \times (\frac{r-m}{s} + 1) \times k$ . It can be found that the length of the input data is compressed from  $r$  to  $(\frac{r-m}{s} + 1)$ . In addition, the data dimension changes from  $d$  (number of sensors) to  $k$  (number of filters), where  $k$  is much larger than  $d$ . This means that the data becomes more informative. In other word, the convolutional neural network can be treated as a local feature learned which is able to get more informative representations and preserve the temporal information from raw environmental sensor data.

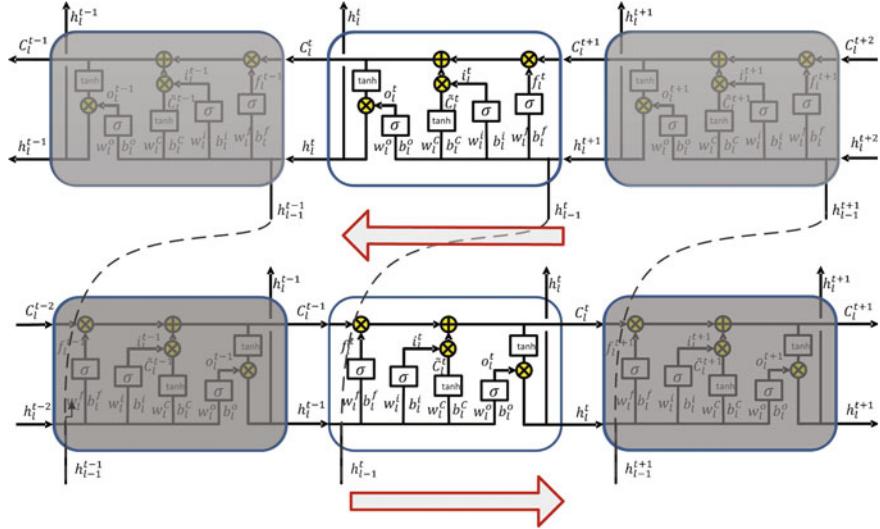
### 3.3 Deep Bi-directional LSTM

Recurrent Neural Network (RNN) is widely used for the modeling of time series data thanks to its strong sequential modeling capacity. However, the conventional RNN

often has the problem of gradient vanishing or exploding during training. This dramatically influence the performance of RNN on modeling long-term dependencies in time-series data [2]. To solve this issue, the authors in [17] proposed a new architecture, named LSTM, which attempts to use some gates to control the information for preserving or discarding, such that it is able to capture long-term dependencies of the sequence. The LSTM network has been successfully employed in a number of important and challenging tasks, e.g., activity recognition [9, 10] and natural language processing [34]. The conventional LSTM only considers the sequential information in one direction, that is the forward direction. This is not adequate for sequential modeling of environmental sensor data. The future information may also be useful. To consider both the future and past contexts for occupancy inference, we adopt the BLSTM which contains a forward layer and a backward layer to process sequential data in the forward and backward directions.

Recently, deep structures have achieved great success in representation learning [16]. The Deep Bi-directional LSTM (DBLSTM) which stacked multiple BLSTM layers is adopted in this study to encode the temporal dependencies and learn high-level features from the sequential local features extracted by the convolutional neural network. In addition to that, the DBLSTM is able to make the inputs to propagate through time and space (layers), simultaneously, such that, the model parameters are able to distribute over layers instead of enlarging memory size of the network. This will result a more efficient non-linear operation of the data and is also the ultimate purpose for stacking multiple layers in deep learning [16]. Figure 3 illustrates a hidden layer  $l$  at time step  $t - 1$ ,  $t$  and  $t + 1$  of the DBLSTM network, where the arrows pointing to the left and right denote the backward and forward operations respectively. Here, the forward operation from time step  $t - 1$  to  $t$  is to capture the past information, and the backward operation from time step  $t + 1$  to  $t$  is to model the future information. We use one hidden layer  $l$  at time step  $t$  as an example to show the detailed operation of the DBLSTM network. Assume that  $h_{l-1}^t$  is the hidden state,  $C_l^{t-1}$  is the memory cell state,  $w_l^f$ ,  $w_l^i$ ,  $w_l^C$  and  $w_l^o$  are the weights,  $b_l^f$ ,  $b_l^i$ ,  $b_l^C$  and  $b_l^o$  are the biases, and  $\sigma(\cdot)$  denotes the sigmoid activation function. The forward process shown as  $\rightarrow$  and the backward process shown as  $\leftarrow$  can be formulated as follows:

$$\begin{aligned}
 \vec{f}_l^t &= \sigma \left( \vec{w}_l^f [\vec{h}_l^{t-1}, \vec{h}_{l-1}^t] + \vec{b}_l^f \right) \\
 \vec{i}_l^t &= \sigma \left( \vec{w}_l^i [\vec{h}_l^{t-1}, \vec{h}_{l-1}^t] + \vec{b}_l^i \right) \\
 \vec{C}_l^t &= \tanh \left( \vec{w}_l^C [\vec{h}_l^{t-1}, \vec{h}_{l-1}^t] + \vec{b}_l^C \right) \\
 \vec{C}_l^t &= \vec{f}_l^t * \vec{C}_l^{t-1} + \vec{i}_l^t * \vec{C}_l^t \\
 \vec{o}_l^t &= \sigma \left( \vec{w}_l^o [\vec{h}_l^{t-1}, \vec{h}_{l-1}^t] + \vec{b}_l^o \right) \\
 \vec{h}_l^t &= \vec{o}_l^t * \tanh (\vec{C}_l^t)
 \end{aligned} \tag{5}$$



**Fig. 3** Structure of DBLSTM

$$\begin{aligned}
 \overleftarrow{f}_l^t &= \sigma \left( \overleftarrow{w}_l^f [\overleftarrow{h}_l^{t+1}, \overleftarrow{h}_{l-1}^t] + \overleftarrow{b}_l^f \right) \\
 \overleftarrow{i}_l^t &= \sigma \left( \overleftarrow{w}_l^i [\overleftarrow{h}_l^{t+1}, \overleftarrow{h}_{l-1}^t] + \overleftarrow{b}_l^i \right) \\
 \overleftarrow{\tilde{C}}_l^t &= \tanh \left( \overleftarrow{w}_l^c [\overleftarrow{h}_l^{t+1}, \overleftarrow{h}_{l-1}^t] + \overleftarrow{b}_l^c \right) \\
 \overleftarrow{C}_l^t &= \overleftarrow{f}_l^t * \overleftarrow{C}_l^{t+1} + \overleftarrow{i}_l^t * \overleftarrow{\tilde{C}}_l^t \\
 \overleftarrow{o}_l^t &= \sigma \left( \overleftarrow{w}_l^o [\overleftarrow{h}_l^{t+1}, \overleftarrow{h}_{l-1}^t] + \overleftarrow{b}_l^o \right) \\
 \overleftarrow{h}_l^t &= \overleftarrow{o}_l^t * \tanh \left( \overleftarrow{C}_l^t \right)
 \end{aligned} \tag{6}$$

The final output of the  $l$ -th hidden layer at time  $t$  of the DBLSTM network is a concatenation of the forward and backward layers, which can be expressed as

$$h_l^t = \overrightarrow{h}_l^t \oplus \overleftarrow{h}_l^t \tag{7}$$

where  $\overrightarrow{h}_l^t$  can update the current hidden state by using the past information, that is the time from 1 to  $t - 1$ , and  $\overleftarrow{h}_l^t$  can update the current hidden state by using the future information, that is the time from  $t + 1$  to  $r$ .

### 3.4 Occupancy Inference Layers

The outputs of the DBLSTM network are high-level features which will be fed into some fully connected layers to get more abstract representations. The expression of the fully connected layers can be shown as:

$$\mathbf{o}^i = g(\alpha_i \mu^i + \beta_i) \quad (8)$$

where  $\mu^i$  and  $\mathbf{o}^i$  are the input and output of the  $i$ -th fully connected layer respectively,  $\alpha_i$  and  $\beta_i$  are the weights and bias respectively, and  $g(\cdot)$  is the activation function. We choose the activation function of ReLU in this study. Suppose that we have stacked  $c$  fully connected layers, the output of the last fully connected layer, known as  $\mathbf{o}^{c-1}$ , is the final representation of the input data. The final feature representations are fed into a softmax classification layer to obtain the occupancy.

### 3.5 Training Process of the CDBLSTM

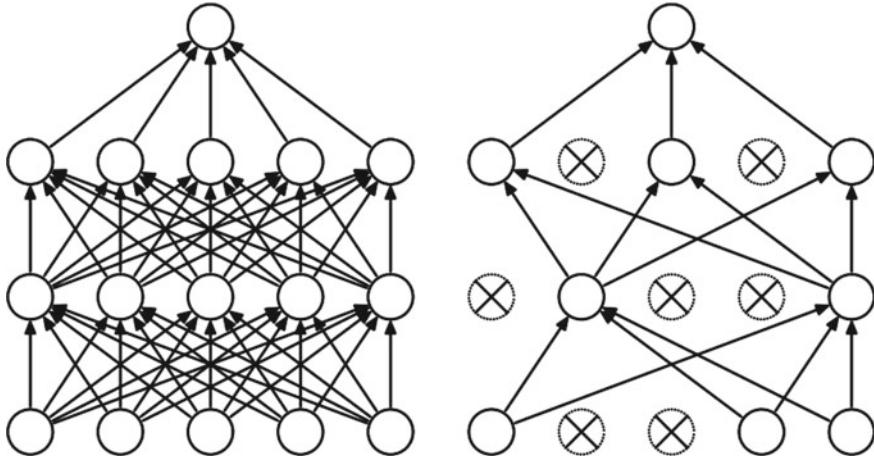
With the outputs of the CDBLSTM and the true labels (occupancy ranges), the errors can be calculated over all the training data, and then error gradients will be derived and back-propagated to adjust model parameters for the training of CDBLSTM [37]. More precisely, given training data with the true occupancy levels, the network outputs can be calculated. Then, the cross-entropy losses can be derived based on the network outputs and true occupancy levels. Next, we can get the error gradients to back-propagate for the adjustment of model parameters via some gradient based optimization algorithms. In this study, we adopt the popular optimization method of RMSprop [35]. Precisely, given  $\theta_t$  the parameter for optimization, and  $L(\theta_t)$  the loss function, the parameter update of  $\theta_{t+1}$  by using the optimization method of RMSprop can be calculated as:

$$g_{t+1} = \gamma g_t + (1 - \gamma) \nabla L(\theta_t)^2 \quad (9)$$

$$\theta_{t+1} = \theta_t - \frac{\eta \nabla L(\theta_t)}{\sqrt{g_{t+1}} + \epsilon} \quad (10)$$

where  $g_t$  is a moving average of the squared gradient at time step  $t$ , and the learning rate  $\eta$ , the parameter  $\gamma$  and the decaying rate  $\epsilon$  are chosen to be 0.001, 0.9 and 0, respectively.

In order to alleviate the overfitting problem, we use the technique of dropout. By using dropout, we will randomly mask parts of the hidden nodes with probability  $p$  during training. Figure 4 illustrate the operation of dropout. During model training, a thinned architecture will be preserved and trained each time. Given a network containing  $n$  nodes with a dropout probability of  $p$  equaling to 0.5, the network could be treated as an ensemble of  $2^n$  thinned networks. Due to the shared structure



**Fig. 4** The operation of dropout. **Left:** the network without dropout; **Right:** the network after dropout. Crossed nodes have been dropped during model training [33]

of these thinned networks, the number of parameters will remain the same. During testing, the dropout will be switched off and all the network nodes will take effect for model outputs, which is similar to an ensemble of some distinct thinned networks. In other words, the dropout is used to enlarge training data size. In each training iteration, random masking will also create some variants into data, which will make the trained network more robust. The dropout technique has been shown to be effective for preventing Overfitting [33]. Therefore, in this study, we leverage on one dropout layer between the DBLSTM and the first fully-connected layer and another dropout layer between the two fully connected layers, where the masking probabilities are chosen to be 0.5 and 0.3 respectively.

## 4 Evaluation Results

In this section, we firstly introduce the data acquisition process. Then, evaluation setup and experimental results are presented. After that, the generalization performance of the CDBLSTM is analyzed by randomly selecting the data for training and testing. Finally, to further demonstrate the performance of CDBLSTM for building occupancy inference using environmental sensors, we demonstrate additional results of the CDBLSTM using data collected from another environment, i.e., a tutorial room.

## 4.1 Data Collection

The sensor data of CO<sub>2</sub>, temperature, air pressure and humidity have been collected from a research lab at a university campus. The lab has an office area which contains 24 cubicles and 11 open seats. Generally, nine postgraduate students and eleven research staffs will work at the office area. Besides, the lab also has six PCs for undergraduate students on their final year projects and five PCs for other students. It is well known that identifying the exact occupancy (number) is very challenging and may require to use some high-cost sensors in a crowded space. Here, instead of estimating the exact occupancy, we divide the exact occupancy into ranges of zero, low, medium and high. These occupancy ranges are enough for common building control and scheduling systems [18]. To make the four ranges balanced, which will maximize the impact of state changes, we define the low occupancy as 1–6 subjects, the medium occupancy as 7–14 subjects, and the high occupancy as larger than 14 subjects.

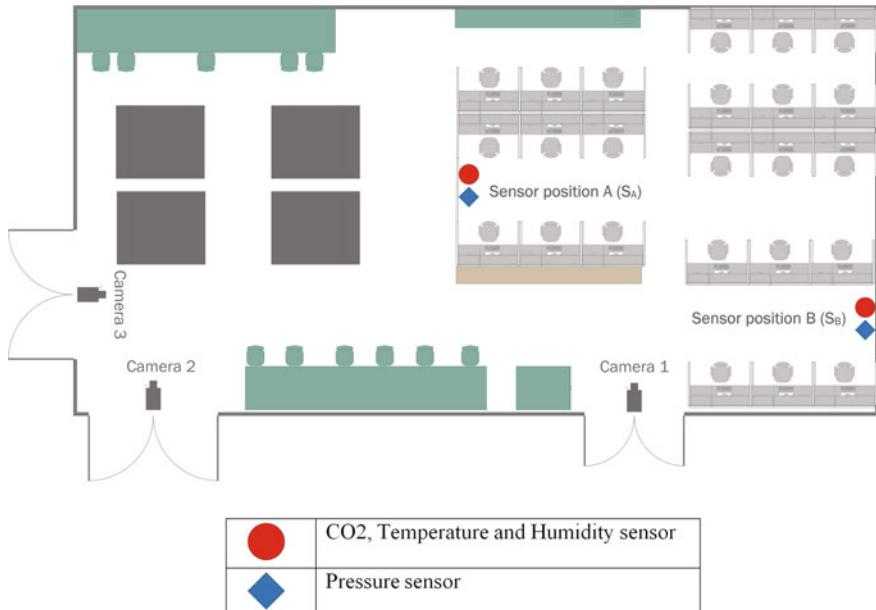
We measure pressure level by leveraging on Lutron MHB-382SD sensor, and CO<sub>2</sub>, temperature, and relative humidity by using the CL11 sensor from Rotronic. The sampling frequency is one sample per minute for both sensors. During data collection, we firstly stored the data in the sensor internal memory and then transmitted to a PC by using a USB cable. Note that, the area is air-conditioned by the conventional Variable Air Volume and Active Chilled Beam systems, and is ventilated by Air Handling Unit (AHU) that will constantly provide fresh air.

Table 1 shows the accuracy and resolution of the sensors. During experiments, we attach the sensors on supporters with a height of 1.1 m from the ground. Figure 5 illustrates the layout of the space which has a size of 20 m × 9.3 m × 2.6 m. We apply two pairs of sensors in this space. Here, the placements of sensors are intuitively selected considering occupant density. To get ground truth occupancy, we deploy three IP cameras at each door to record occupant movements. Then, the true occupancy is counted manually with the help of motion detection software which is able to take pictures when occupants move. The entire space contains three doors. The main door (placement of camera 1) connects the space with the office area for administrative staffs. Another door which locates at camera 2 in Fig. 5 opens to a lab space. And the third door is always closed. Note that, all windows are closed, due to the operation of air-conditioning and ventilation systems.

Totally, we collected 31 days of data in workdays, where the first 26 days of data are utilized for model training and the rest 5 days of data is utilized for model testing. Since building control systems are with slow response, a resolution of 15-min is enough for occupancy estimation [39]. But the original sensor data and occupancy have a resolution of 1 min, we firstly transfer them into a 15-min resolution by using the simple averaging. Note that, the number of occupants are an integer value, so that a rounding operation is conducted after the use of averaging on original occupancy.

**Table 1** The accuracy and resolution the sensors

Sensor	Environmental parameter	Resolution	Accuracy
Rotronic CL11	CO <sub>2</sub>	1 ppm	±5% of the measured value
	Temperature	0.05 °C	±0.3 °K
	Humidity	0.1% RH	<2.5% RH
Lutron MHB-382SD	Pressure	0.1 hPa	±2 hPa

**Fig. 5** Layout of the research lab

#### 4.2 Evaluation Setup

To evaluate the performance of CDBLSTM, a comparison has been made between the CDBLSTM and some state-of-the-arts including the HMM approach with the information gain based feature selection of some statistical handcrafted features (Dong's method) [13], the DT with raw data for features (Yang's method) [38], the ELM with the wrapper based feature selection of some statistical handcrafted features (Masood's method) [30], and the LDA with raw data for features (Candanedo's method) [3].

The DBLSTM without the convolutional network for local sequential feature extraction is also implemented for comparison. Since we choose the resolution to be 15-min and the sampling frequency of sensors is 1-min, the length of the input

sequence  $r$  is 15. With 2 pairs of sensors shown in Fig. 5, the total number of sensors  $d$  is 8. Hence, the input is with a dimension of  $15 \times 8$  for environmental sensor based occupancy estimation. We use cross-validation with the training data to choose proper hyperparameters for all the approaches. Specifically, the DBLSTM consists of three BLSTM layers with hidden nodes of 24, 75 and 100. Then, two fully connected layers with hidden nodes of 150 and 100 are adopted. For the CDBLSTM approach, the window size, the pooling size and the number of filters are chosen to be 3, 2, 100, respectively. The CDBLSTM contains three BLSTM layers with hidden size to be 100, 150 and 200. The two fully-connected layers have 200 and 300 hidden nodes. The implementation of the deep algorithms, i.e., CDBLSTM and DBLSTM, is under Keras. The other shallow algorithms are performed using Matlab.

Here, occupancy estimation is regarded as a typical classification problem. Hence, the criterion of classification accuracy can be adopted for model performance evaluation. Besides, we use another widely used evaluation criterion of Normalized Root Mean Square Error (NRMSE) which will show the range of classification errors [38]. As we all know, the absence and presence are of great significance for building control systems, especially the light control system [32], the detection accuracy of the two states is also analyzed.

### 4.3 Evaluation Results

The evaluation results for different methodologies under the defined three evaluation criteria are shown in Table 2. Candanedo's and Yang's approaches which applied the raw data as features performs the worst. Note that Candanedo et al. [3] and Yang et al. [38] used many sensors in their works to guarantee the satisfactory performance, which is not practical due to the high cost and the inconvenience caused by constant maintenance. Masood's and Dong's approaches performs better than Candanedo's and Yang's approaches, due to the use of statistical features instead of raw data for features. These results clearly show that feature extraction is compulsory and useful, especially with limited sensors. Since Masood's and Dong's methods used

**Table 2** The Evaluation results of different methods under the three evaluation criteria. P/A represents Presence/Absence

Criterion	Dong's [13]	Yang's [38]	Masood's [30]	Candanedo's [3]	DBLSTM	CDBLSTM
Classification accuracy (%)	71.46	66.67	72.31	70.21	74.38	<b>76.04</b>
NRMSE	0.1912	0.2509	0.2322	0.2297	0.1574	<b>0.1169</b>
Detection accuracy of P/A (%)	93.13	90.21	92.38	88.54	95.21	<b>95.42</b>

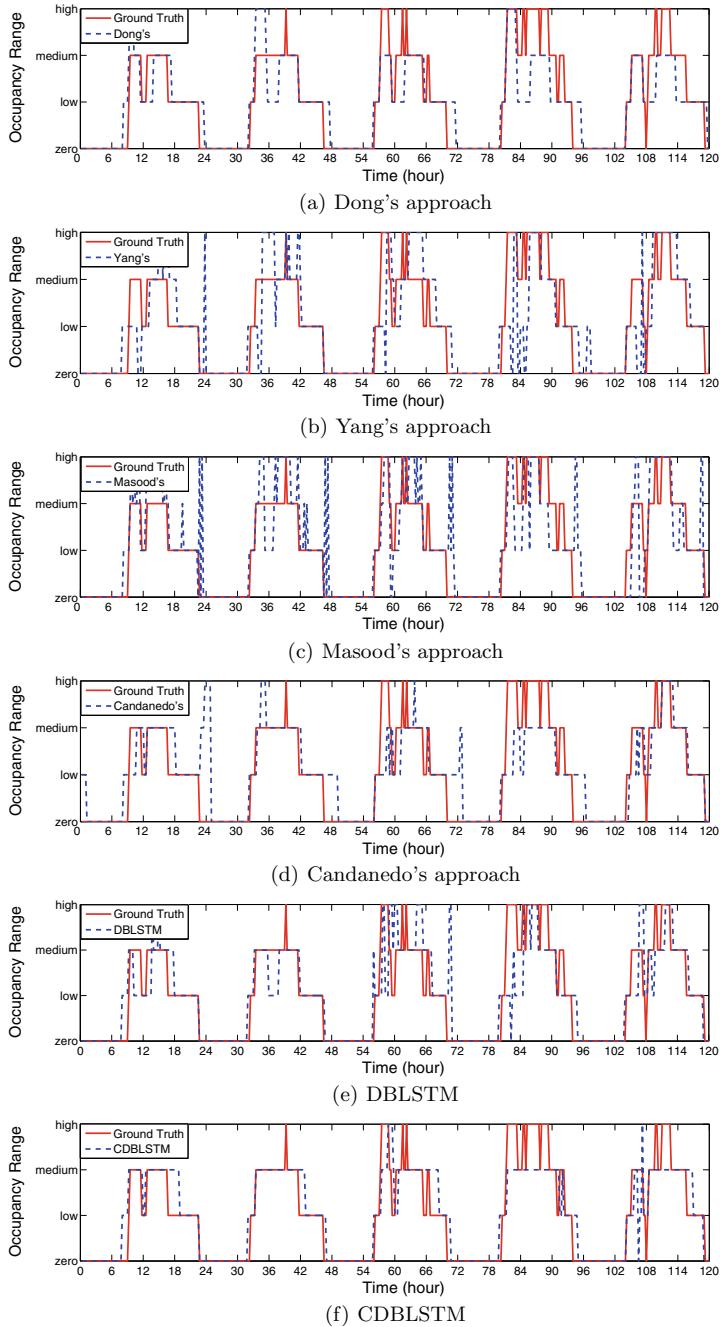
manually extracted features which will inevitably miss useful and implicit features, the performances of these methods are also limited for environmental sensor based human activity recognition.

Owing to the deep structures for feature learning and temporal encoding of the DBLSTM approach, it is able to perform better than all the state-of-the-arts under these three evaluation criteria. With the powerful local feature extractor fulfilled by the convolutional network, the CDBLSTM further enhance the performance of DBLSTM. It outperforms all the approaches where the occupancy estimation accuracy, the NRMSE and the detection accuracy are 76.04%, 0.1169 and 95.42%, respectively.

We also illustrate the occupancy estimation results of all the testing days in Fig. 6, where useful insights can be concluded:

- Candanedo's and Yang's approaches perform worse than other approaches, due to the use of raw data as features. With sensor noise and limited number of sensors, the raw sensor data is not representative for different occupancy levels. The more efficient way is to extract some representative features.
- Since Masood's exhaustively searches the best integration of features with the proposed wrapper method, it overfits on the testing data. Similarly, Dong's method also cannot track occupancy profiles well with the handcrafted features. It can be concluded that handcrafted features lack a clear guideline and will inevitably miss useful and implicit features, which limited the system performance.
- One interesting phenomenon is that the estimated occupancy suddenly increases at midnight for Candanedo's, Masood's and Yang's approaches. By checking the data carefully, it should be caused by a sudden increase of CO<sub>2</sub> data. Then, the recorded video was checked, and we find that one subject sitting near a pair of sensors usually walks around to prepare for leaving at that time. The optimal locations sensors will be considered as one of our future works [20]. Due to the sequential modeling capacity of HMM and the BLSTM structure, Dong's approach, DBLSTM and CDBLSTM can almost immune to this issue caused by the increase of CO<sub>2</sub> data.
- With the deep structure for feature learning and the BLSTM network for temporal encoding, the DBLSTM and CDBLSTM approaches outperforms all the state-of-the-arts.
- Owing to the convolutional network for local feature extraction, the CDBLSTM further enhances the performance of DBLSTM, and its better performance over all methodologies indicates the effectiveness of using CDBLSTM for building occupancy inference based on environmental sensors.

Time complexity is a big concern about deep learning based methods. To show the time complexity of the CDBLSTM, we tested its training and testing time during experiments. Here, the state-of-the-art algorithms all based on manual feature extraction and conventional machine learning algorithms have much smaller training and testing time when compared with CDBLSTM. The CDBLSTM is implemented with a computer which has dual core CPUs of Intel Xeon(R) E5-2697 v2 2.70 GHz and a GPU of NVIDIA Tesla K40c. Its training time is about 16 min and 40 s. Although



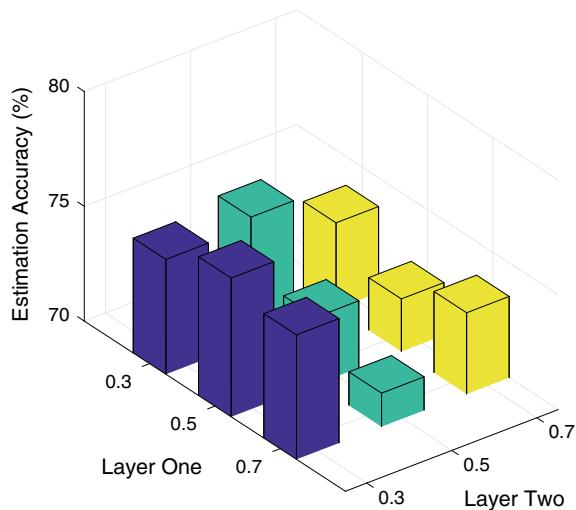
**Fig. 6** The evaluation results of the testing data for all the methodologies [11]

this amount of time for training is large, it is still acceptable because the training only requires to be done once in offline. The testing time of the CDBLSTM for all the samples (480 samples) is 0.35 s. This can be neglected for building control systems with a resolution of 15 min. Hence, we can conclude that the CDBLSTM method can be used for real-time occupancy estimation with environmental sensors.

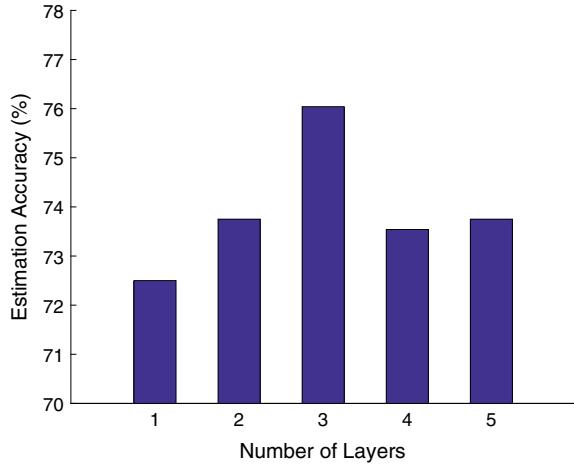
#### 4.4 HyperParameters

Some hyperparameters are crucial for the CDBLSTM approach. Here, the parameters of the masking probabilities of the two dropout layers and the number of hidden layers are investigated. We explored three masking probability levels, including high (0.7), medium (0.5) and low (0.3). Figure 7 demonstrates the occupancy estimation accuracy of the CDBLSTM with different combinations of masking probability. We can find that the CDBLSTM may underfit with a degraded performance when high masking probabilities, such as the combinations of [0.7 0.7], [0.7 0.5], [0.5 0.7] and [0.5 0.5] are used. It is clear that a good selection of this hyperparameter will enhance the performance of CDBLSTM. The number of hidden layers is another key hyperparameter for the model. The estimation performance of the model with distinct number of hidden layers is shown in Fig. 8. When the number of hidden layers increases from 1 to 3, the model performance improves. But, if the number of hidden layers is larger than 4 in this study, the model may overfit, resulting a limited performance.

**Fig. 7** Occupancy Estimation performance of the CDBLSTM with different combinations of masking probability



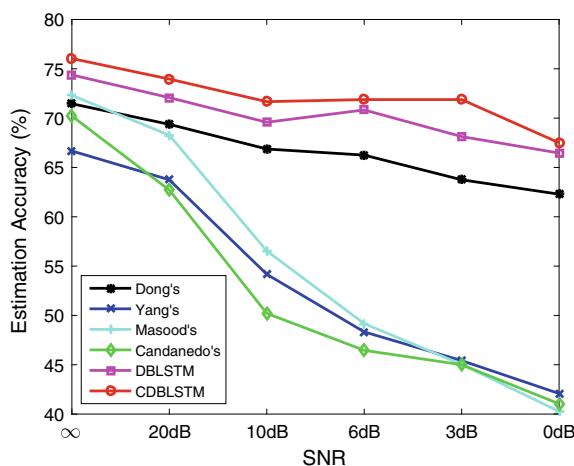
**Fig. 8** Estimation performance of CDBLSTM with varying number of hidden layers



#### 4.5 The Impact of Noise

The CDBLSTM approach is able to almost immune to some abnormal and noisy data as analyzed in Sect. 4.3, due to its ability to consider temporal dependencies in data. In order to explore the robustness of CDBLSTM on noise data, we manually include some noise into the raw sensor data. Figure 9 presents the performance of all the approaches with different noise levels. Note that the signal to noise ratio (SNR) is  $\infty$  when no noise is added. When the SNR decreases (noisier), the performance of all the approaches degrade accordingly. Due to the capability of modeling temporal dependencies in data, the noise impact on the HMM model (Dong's), DBLSTM

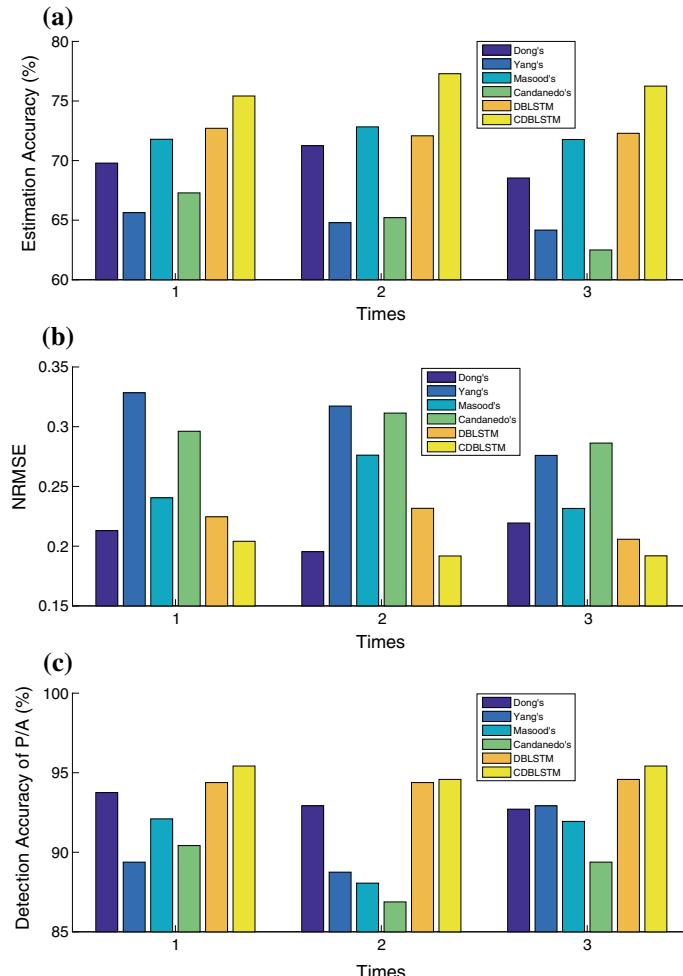
**Fig. 9** Estimation performance with varying SNR



and CDBLSTM is smaller, which is consistent with the previous conclusion. The evaluation manifests that the CDBLSTM approach is robust against the noise in data.

## 4.6 Generalization Performance

In order to verify the generalization performance of the CDBLSTM method, additional experiments are conducted. Specifically, we randomly select five days of data



**Fig. 10** The evaluation results for the analysis of generalization performance **a** estimation accuracy, **b** NRMSE and **c** detection accuracy of P/A

for model testing and the rest for training. Note that, each day of data have equal probability to be chosen as training or testing, that guarantees the indication of the generalization capability of the CDBLSTM approach. We performed three times for the experiments. Figure 10 shows the final results. It can be found that the DBLSTM approach has a better performance than the state-of-the-arts, and CDBLSTM performs the best under the three evaluation criteria. The conclusions are the same as the previous analysis. This clearly manifests the good generalization performance of the CDBLSTM method for environmental sensor based occupancy detection and estimation.

#### **4.7 Additional Evaluation with Data from Another Environment**

To further evaluate the performance of the CDBLSTM, we perform an additional experiment with the data collected from a tutorial room. Totally, we collected fourteen workdays of data for evaluation, where we randomly choose eleven days of data for training and the rest for testing. A more comprehensive illustration of data is presented in [30]. The evaluation results of all the approaches is shown in Table 3. It can be found that all the approaches perform worse in this scenario. The reason is that we only deployed one pair of sensors in this large environment. To enhance the performance, more sensors should be deployed. In this evaluation, we can get the same conclusion. The DBLSTM outperforms all the state-of-the-arts. The CDBLSTM performs the best. This further manifests the effectiveness and robustness of the CDBLSTM approach for environmental sensor based building occupancy estimation.

**Table 3** Evaluation results in the tutorial room

Criterion	Dong's [13]	Yang's [38]	Masood's [30]	Candanedo's [3]	DBLSTM	CDBLSTM
Estimation accuracy (%)	57.78	54.44	54.22	55.56	58.89	<b>65.56</b>
NRMSE	0.3768	0.3201	0.3214	0.3296	0.2676	<b>0.2383</b>
Detection accuracy of P/A (%)	70.00	78.89	85.22	78.89	85.56	<b>87.78</b>

## 5 Conclusion

This chapter introduces a deep learning algorithm, termed Convolutional Deep Bi-directional Long Short-Term Memory (CDBLSTM), for environmental sensor based occupancy inference in buildings. The CDBLSTM consists of a convolutional network for sequential local feature extraction from the raw environmental sensor data and a DBLSTM for temporal coding and feature learning. To verify the performance of CDBLSTM, we perform experiments in a research lab environment and compare with some existing approaches and the DBLSTM method without the convolutional operation. The results indicate that DBLSTM outperforms the state-of-the-arts and CDBLSTM has the best performance, which indicates the merits of the convolutional network and the DBLSTM structure for temporal encoding and feature learning. We also test some hyperparameters of the CDBLSTM with a conclusion that a proper selection of model hyperparameters will boost the performance of CDBLSTM. Then, the impact of noise on model performance is evaluated. The results manifests that the CDBLSTM is able to alleviate the noise effect due to its unique structure. After that, we test the generalization performance of the CDBLSTM by randomly selecting data for training and testing. We can obtain the same conclusion in this scenario. Finally, we perform an additional test in a tutorial room. Similarly, the CDBLSTM achieves a superior performance over all the other methodologies.

## References

1. Balaji, B., Xu, J., Nwokafor, A., Gupta, R., Agarwal, Y.: Sentinel: occupancy based HVAC actuation using existing WiFi infrastructure within commercial buildings. In: Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, p. 17. ACM (2013)
2. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **5**(2), 157–166 (1994)
3. Candanedo, L.M., Feldheim, V.: Accurate occupancy detection of an office room from light, temperature, humidity and CO<sub>2</sub> measurements using statistical learning models. *Energy Build.* **112**, 28–39 (2016)
4. Chen, Z., Jiang, C.: Building occupancy modeling using generative adversarial network. *Energy Build.* **174**, 372–379 (2018)
5. Chen, Z., Jiang, C., Xie, L.: Building occupancy estimation and detection: a review. *Energy Build.* (2018)
6. Chen, Z., Masood, M.K., Soh, Y.C.: A fusion framework for occupancy estimation in office buildings based on environmental sensor data. *Energy Build.* **133**, 790–798 (2016)
7. Chen, Z., Soh, Y.C.: Modeling building occupancy using a novel inhomogeneous Markov chain approach. In: 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp. 1079–1084. IEEE (2014)
8. Chen, Z., Xu, J., Soh, Y.C.: Modeling regular occupancy in commercial buildings using stochastic models. *Energy Build.* **103**, 216–223 (2015)
9. Chen, Z., Zhang, L., Cao, Z., Guo, J.: Distilling the knowledge from handcrafted features for human activity recognition. *IEEE Trans. Ind. Inform.* (2018)
10. Chen, Z., Zhang, L., Jiang, C., Cao, Z., Cui, W.: WiFi CSI based passive human activity recognition using attention based BLSTM. *IEEE Trans. Mob. Comput.* (2018)

11. Chen, Z., Zhao, R., Zhu, Q., Masood, M.K., Soh, Y.C., Mao, K.: Building occupancy estimation with environmental sensors via CDBLSTM. *IEEE Trans. Ind. Electron.* **64**(12), 9549–9559 (2017)
12. Chen, Z., Zhu, Q., Masood, M.K., Soh, Y.C.: Environmental sensors-based occupancy estimation in buildings via IHMM-MLR. *IEEE Trans. Ind. Electron.* **13**(5), 2184–2193 (2017)
13. Dong, B., Andrews, B., Lam, K.P., Höynck, M., Zhang, R., Chiou, Y.S., Benitez, D.: An information technology enabled sustainability test-bed (ITEST) for occupancy detection through an environmental sensing network. *Energy Build.* **42**(7), 1038–1046 (2010)
14. Frodl, R., Tille, T.: A high-precision NDIR gas sensor for automotive applications. *IEEE Sens. J.* **6**(6), 1697–1705 (2006)
15. Gong, Y., Wang, L., Guo, R., Lazebnik, S.: Multi-scale orderless pooling of deep convolutional activation features. In: European Conference on Computer Vision, pp. 392–407. Springer (2014)
16. Hinton, G.E.: Learning multiple layers of representation. *Trends Cogn. Sci.* **11**(10), 428–434 (2007)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
18. Iglesias, F., Palensky, P.: Profile-based control for central domestic hot water distribution. *IEEE Trans. Ind. Inform.* **10**(1), 697–705 (2014)
19. Jiang, C., Chen, Z., Png, L.C., Bekiroglu, K., Srinivasan, S., Su, R.: Building occupancy detection from carbon-dioxide and motion sensors. In: International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 931–936 (2018)
20. Jiang, C., Chen, Z., Su, R., Soh, Y.C.: Group greedy method for sensor placement. *IEEE Trans. Signal Process.* **67**(9), 2249–2262 (2019)
21. Jiang, C., Masood, M.K., Soh, Y.C., Li, H.: Indoor occupancy estimation from carbon dioxide concentration. *Energy Build.* **131**, 132–141 (2016)
22. Jin, M., Jia, R., Spanos, C.J.: Virtual occupancy sensing: using smart meters to indicate your presence. *IEEE Trans. Mob. Comput.* **16**(11), 3264–3277 (2017)
23. Labeodan, T., Zeiler, W., Boxem, G., Zhao, Y.: Occupancy measurement in commercial office buildings for demand-driven control applications a survey and detection system evaluation. *Energy Build.* **93**, 303–314 (2015)
24. Leephakpreeda, T.: Adaptive occupancy-based lighting control via grey prediction. *Build. Environ.* **40**(7), 881–886 (2005)
25. Li, N., Calis, G., Becerik-Gerber, B.: Measuring and monitoring occupancy with an RFID based system for demand-driven HVAC operations. *Autom. Constr.* **24**, 89–99 (2012)
26. Liu, H., Motoda, H.: Feature Extraction, Construction and Selection: A Data Mining Perspective. Springer Science & Business Media (1998)
27. Liu, P., Nguang, S.K., Partridge, A.: Occupancy inference using pyroelectric infrared sensors through hidden Markov models. *IEEE Sens. J.* **16**(4), 1062–1068 (2016)
28. Mantovani, G., Ferrarini, L.: Temperature control of a commercial building with model predictive control techniques. *IEEE Trans. Ind. Electron.* **62**(4), 2651–2660 (2015)
29. Masood, M.K., Jiang, C., Soh, Y.C.: A novel feature selection framework with hybrid feature-scaled extreme learning machine (HFS-ELM) for indoor occupancy estimation. *Energy Build.* **158**, 1139–1151 (2018)
30. Masood, M.K., Soh, Y.C., Chang, V.W.C.: Real-time occupancy estimation using environmental parameters. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2015)
31. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010)
32. Parise, G., Martirano, L., Cecchini, G.: Design and energetic analysis of an advanced control upgrading existing lighting systems. *IEEE Trans. Ind. Appl.* **50**(2), 1338–1347 (2014)
33. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014)

34. Sundermeyer, M., Schlüter, R., Ney, H.: LSTM neural networks for language modeling. In: Thirteenth Annual Conference of the International Speech Communication Association (2012)
35. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA Neural Netw. Mach. Learn. **4**(2) (2012)
36. Tran, D., Tan, Y.K.: Sensorless illumination control of a networked led-lighting system using feedforward neural network. IEEE Trans. Ind. Electron. **61**(4), 2113–2121 (2014)
37. Williams, D., Hinton, G.: Learning representations by back-propagating errors. Nature **323**(6088), 533–538 (1986)
38. Yang, Z., Li, N., Becerik-Gerber, B., Orosz, M.: A systematic approach to occupancy modeling in ambient sensor-rich buildings. Simulation **90**(8), 960–977 (2014)
39. Yu, Z., Jia, L., Murphy-Hoye, M.C., Pratt, A., Tong, L.: Modeling and stochastic control for home energy management. IEEE Trans. Smart Grid **4**(4), 2244–2255 (2013)
40. Zhu, Q., Chen, Z., Masood, M.K., Soh, Y.C.: Occupancy estimation with environmental sensing via non-iterative LRF feature learning in time and frequency domains. Energy Build. **141**, 125–133 (2017)
41. Zimmermann, L., Weigel, R., Fischer, G.: Fusion of nonintrusive environmental sensors for occupancy detection in smart homes. IEEE Internet Things J. **5**(4), 2343–2352 (2018)
42. Zou, J., Zhao, Q., Yang, W., Wang, F.: Occupancy detection in the office by analyzing surveillance videos and its application to building energy conservation. Energy Build. **152**, 385–398 (2017)

# Index

## A

Activation functions, 1–28, 45, 71, 87, 93, 113–117, 120, 123, 139, 265–267, 274, 276, 277, 280, 283–288, 308, 309, 323, 340, 342, 344  
Adaptive activation functions, 20  
Adversarial defenses, 32, 58  
Adversarial examples, 31–33, 35–37, 39, 40, 43–45, 47, 49, 51–54, 58–61  
Adversarial training, 52, 56, 58–61  
Application, 6, 10, 36, 61, 79, 81, 88, 99, 104, 108, 114, 115, 125, 126, 129, 136, 148, 151, 159, 161, 165–167, 177, 193, 196, 199, 202, 208, 214, 216, 218, 224, 226, 232, 234, 235, 240, 260, 263, 265, 267, 269, 270, 299–301, 303, 305, 306, 308, 311, 321, 322, 326, 327, 329  
Autonomous vehicle, 157, 158, 165, 166, 168, 172, 177, 183, 185, 186, 189, 195, 197, 202

## B

Biometrics, 299–301, 303–305, 321–326, 328–330  
Building occupancy estimation, 336, 354

## C

Classification, 1–3, 5, 6, 25, 32–34, 36, 37, 39, 42, 48–55, 57, 61, 69, 73, 77, 78, 114, 161, 168, 193–197, 199, 214, 218, 226, 231, 232, 234–236, 239,

244, 246, 249, 251–255, 260, 261, 276, 278, 285, 288, 301–303, 307, 308, 311, 313–319, 329, 338, 340, 344, 348

Convolutional Deep Bi-directional Long Short-TermMemory (CDBLSTM), 335–339, 344, 345, 347–349, 351–355

Convolutional Neural Networks (CNNs), 12, 14, 17, 25, 49, 50, 59, 79–81, 113–115, 129, 136–138, 140, 147, 148, 150, 157, 167, 168, 195, 196, 199, 215, 231, 232, 265, 283–288, 292, 301, 308–310, 322, 329, 335, 337, 338, 340–342

## D

Data augmentation, 167, 232, 237, 267  
Data distribute service, 158  
Deep learning, 103, 105, 106, 108, 110, 111, 113, 117–125, 158, 159, 167, 168, 173, 177, 184–186, 189, 231, 232, 263–265, 274, 277, 280, 283, 285, 286, 291, 292  
Dual-task, 300, 311–317, 319, 320, 322, 329, 330

## E

Energy consumption prediction, 106, 336  
Environmental sensors, 335–338, 345, 349, 351  
Exploration geophysics, 130, 144, 154

**G**

Gait analysis, 299–301, 303–308, 310–312, 321

**H**

High dynamic Range imaging, 189

**I**

Intelligent surveillance systems, 264  
Inverse problems, 145

**J**

Jellyfish, 193, 195, 197, 213, 214, 216, 218–220, 224–226

**L**

Learning deep neural networks, 1  
Load forecasting, 67, 68, 70, 103–121, 125, 126

**M**

Machine Learning (ML), 5, 14, 24, 32, 33, 49, 56, 71, 88, 98, 105, 106, 110, 112, 113, 121, 124, 125, 130, 159, 167, 195, 199, 234, 292, 300, 301, 307, 311, 314–317, 319, 321, 323, 325, 329, 330, 336–338, 349

Marine, 193–195, 226, 234

**N**

Neural Networks (NNs), 1–4, 6, 9, 10, 15, 16, 21, 27, 28, 43, 45, 49, 50, 56, 57, 70, 71, 113–115, 157, 167, 168, 196, 217, 226, 231, 232, 234, 264, 265, 270, 273, 274, 277, 278, 280, 283–287, 308, 310

**O**

Object detection, 43, 167, 168, 173, 185, 195–197, 214, 218, 226

**P**

Perturbation analysis, 31, 35  
Posidonia oceanica, 193, 195, 197, 204

**R**

Regression, 2, 3, 5, 6, 42, 43, 70, 71, 77, 78, 81–84, 87, 88, 99, 111–113, 136, 145, 185, 338  
Re-identification, 263–265, 274, 277, 280, 281, 283, 284, 289, 291, 292  
ReLU, 10–12, 93, 123, 196, 240, 241, 244, 265, 267, 276, 277, 283–287, 340, 344  
Renewable energy, 67, 69, 85, 104, 106, 107, 125  
Representation learning, 67, 71–73, 78, 81, 85, 98, 99, 342

**S**

Seismic imaging, 129, 132, 154  
Self Learnable Activation Function (SLAF), 1, 20, 21, 23–28  
Semantic segmentation, 195, 196, 199, 200, 226  
Smart grids, 71, 103–108, 118, 119, 125, 126  
Statistical learning, 33, 56, 61, 124, 136

**T**

Time series, 2, 14, 68–71, 79–85, 87, 88, 95, 96, 98, 99, 106, 111–118, 121, 124, 337, 341  
Tomography, 129–133, 135–137, 151, 154, 304, 311, 314  
Traffic light recognition, 157–160, 163, 166, 167, 170

**V**

Vehicle signal recognition, 157–161, 184, 185  
Video surveillance, 263, 264, 292, 301, 302