

POZNAN UNIVERSITY OF TECHNOLOGY



PHD DISSERTATION

---

**Restricted Boltzmann Machine  
as a binary image descriptors processor  
and its application in a mobile robot  
for scene recognition**

---

Ograniczona maszyna Boltzmann jako procesor binarnych  
deskryptorów obrazu oraz jej aplikacja w robocie mobilnym  
do celów przetwarzania wizji

---

*Author:*  
Szymon Sobczak, MSc

*Supervisors:*  
Dariusz Pazderski, PhD, DSc  
Rafał Kapela, PhD

30/05/2023



*First, I would like to express  
my sincere gratitude to those without whom  
finishing this work would not have been possible.*

*I am deeply grateful to Dariusz Pazderski, PhD, DSc,  
for his guidance, patience,  
and support during my PhD studies  
and throughout the writing of this dissertation.*

*I would also like to acknowledge Rafał Kapela, PhD,  
for inspiring me to conduct this research  
and for his invaluable assistance and advice.*

*Furthermore, I extend my heartfelt appreciation  
to my family for their  
unwavering support and understanding,  
particularly my wife, Milena,  
for her patience and encouragement,  
and to my daughter, Jagna,  
who has been a constant source of joy and motivation.*



---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Machine vision . . . . .	5
1.2	Neural networks . . . . .	6
1.2.1	Basic concepts of neural networks . . . . .	6
1.2.2	Neural networks in vision . . . . .	12
1.2.3	Neural networks in robotics . . . . .	14
1.2.4	Deep neural networks complexity . . . . .	16
1.3	Research problems and proposed solutions . . . . .	17
1.4	Dissertation scope and structure . . . . .	19
<b>2</b>	<b>Image feature engineering</b>	<b>21</b>
2.1	Convolutional layers . . . . .	22
2.2	Unsupervised feature extractors and autoencoders . . . . .	26
2.3	Energy-based models . . . . .	27
2.4	Feature descriptors . . . . .	31
2.4.1	Binary descriptors . . . . .	31
2.4.2	Local Binary Pattern . . . . .	32
2.4.3	Real-valued feature descriptors . . . . .	35
2.4.4	Feature descriptors aggregation . . . . .	36
2.5	Enhanced Local Binary Pattern - colour LBP8 . . . . .	38
<b>3</b>	<b>Restricted Boltzmann Machine</b>	<b>43</b>
3.1	Overview of an RBM . . . . .	43
3.2	Training an RBM . . . . .	52
3.3	Deep Boltzmann Machines . . . . .	59
<b>4</b>	<b>Image processing and classification with binary descriptors and Restricted Boltzmann Machines</b>	<b>61</b>
4.1	The RBM and the CLBP - author's implementation . . . . .	62
4.2	The RBM as a binary descriptors processor and aggregator, CD feature space . . . . .	64
4.3	The RBM as a binary descriptors processor, hidden layer fea- ture space . . . . .	68
4.4	RBMs for visual data comparison . . . . .	74

---

4.5	RBM for input data denoising . . . . .	78
<b>5</b>	<b>Experimental research</b>	<b>81</b>
5.1	Image classification with RBM's hidden units as a feature space	88
5.1.1	LBP8-RBM - MNIST dataset experiments . . . . .	88
5.1.2	Colour Local Binary Pattern . . . . .	95
5.1.3	CLBP-RBM - CIFAR-10, STL-10, Concrete Cracks experiments . . . . .	96
5.2	CLBP-RBM for visual data comparison . . . . .	106
5.3	CLBP-RBM for visual input denoising . . . . .	109
5.3.1	Binary vectors denoising . . . . .	109
5.3.2	Adversarial attack . . . . .	113
5.3.3	Overall distortion sensitivity of CLBP-RBM preprocessing . . . . .	114
5.4	Image classification with Contrastive Divergence as a feature space . . . . .	123
5.4.1	Binary descriptors aggregation . . . . .	123
5.4.2	CD as an entry for CNN . . . . .	124
<b>6</b>	<b>Mobile robot application</b>	<b>127</b>
6.1	Application goals and experimental setup . . . . .	127
6.2	The architecture of the application . . . . .	129
6.3	Mobile robot kinematics . . . . .	133
6.3.1	Indentification of mobile robot parameters . . . . .	134
6.4	RBM for robot rotation angle detection . . . . .	135
6.5	CLBP-RBM preprocessing for a CNN in mobile robot vision system . . . . .	137
<b>7</b>	<b>Conclusions</b>	<b>157</b>
	<b>Appendix A RBM and CLBP implementation details</b>	<b>161</b>
	<b>Appendix B Training convolutional neural networks</b>	<b>165</b>
	<b>Appendix C Robot's angle deviation measurement with a dedicated vision system</b>	<b>167</b>

---

---

## List of Figures

---

1.1	Diagram of an artificial neuron. . . . .	7
1.2	Diagram of a Multi-Layer Perceptron. . . . .	8
1.3	Diagram of two layers of a Multi-Layer Perceptron. . . . .	9
1.4	Example a of non-linear classification problem with its representation in other spaces (polar and given by the MLP), the classes are marked by two colours, the blue line in the model coordinates shows how the last neuron separates the $(h_1, h_2)$ surface. The second hidden layer of the MLP is not necessary but allows the projection of the hidden space onto 2D surface.	10
1.5	Simplified diagram of a CNN neural network. . . . .	13
1.6	ImageNet top-5 accuracy of some common CNN architectures versus their sizes [19, 33, 34, 35, 36, 37, 38, 58, 59, 60]. . . . .	16
1.7	Simplified diagram of a CNN neural network with unsupervised preprocessing. . . . .	18
2.1	Simplified diagram of a typical image classification pipeline. . . . .	21
2.2	Simplified diagram of a convolution operator on an image (bias and activation function are omitted for simplifying the diagram).	23
2.3	Simplified diagram of a convolution operator with multiple filters on a multi-channels input. . . . .	24
2.4	Simplified diagram of an Autoencoder. . . . .	27
2.5	Simplified diagram of a Hopfield network. . . . .	28
2.6	Theoretical graph of energy function versus states in a Hopfield network. . . . .	30
2.7	Sampling patterns for two example LBP configurations. Green is a centre pixel, yellow denotes neighbouring pixels taken for computing the descriptor, blue pixels are omitted by the descriptor. . . . .	33
2.8	Two bits descriptor distribution for a single colours pair. . . . .	39
2.9	Colour LBP descriptor. . . . .	40
2.10	Example colours with their 8 bits colour descriptors (white rectangles), and RGB hex codes. . . . .	40

---

3.1	Simplified diagram of a Boltzmann Machine. Stochastic dynamics is skipped for simplicity. . . . .	44
3.2	Simplified diagram of a Restricted Boltzmann Machine. . . . .	46
3.3	Simplified diagram of an unrolled Restricted Boltzmann Machine. . . . .	46
3.4	Energy value of a Restricted Boltzmann Machine during its state's updates, normalised to its maximum value. . . . .	52
3.5	Stacking multiple RBMs, DBN/DBM model. . . . .	59
4.1	Image classification with preprocessing with an RBM and Contrastive Divergence feature space. Binary descriptors are gathered from keypoints and CD is averaged for $m$ prior given regions of an image. . . . .	67
4.2	Image classification with preprocessing with an RBM and a Contrastive Divergence feature space. Binary descriptors are gathered densely. . . . .	68
4.3	Image feature extraction with binary descriptors and RBM hidden space. . . . .	69
4.4	Hiperpolic tangent and sigmoid activation functions depending on the $\beta$ scaling factor. . . . .	70
4.5	Image feature extraction with binary descriptors and RBM hidden space with kernel size equal to 2. . . . .	71
4.6	Image classification with preprocessing with RBM hidden space as feature extractor. Binary descriptors are gathered densely. . . . .	72
4.7	Image feature extraction with binary descriptors and mutiple RBM layers. . . . .	74
4.8	Flattening a histogram by adjusting an $\mathbf{m}$ vector (compare various x-scales in both figures). . . . .	76
4.9	Pattern reconstruction in an RBM. . . . .	80
5.1	RBM response time depending on its parameter and size of an input image with a squared shape. . . . .	86
5.2	RBM Response time in milliseconds depending on the size of an input image with a squared shape and number of hidden units. . . . .	87
5.3	Results of experiments on MNIST dataset, a reference network without LBP-RBM achieves accuracy $\approx 0.91 - 0.92$ . . . . .	89
5.4	RBM hyper-parameters tuning on MNIST dataset. . . . .	90
5.5	Validation learning curves with and without RBM preprocessing layer on MNIST dataset. Figure 5.5h contains training curves to emphasize the overfitting. . . . .	94

---



---

5.6	Accuracy on CIFAR-10 dataset depending on the type of descriptor used in preprocessing, "CLBP_2" refers to CLBP without 2 intensity bits. . . . .	96
5.7	Image classification pipeline for CIFAR-10, STL-10 and Concrete Cracks datasets, <b>HS-CNN</b> architecture enhanced with Dropout and GAP. . . . .	96
5.8	RBM performance versus its hyper-parameters. Results are relative to the lowest accuracy and to the highest processing time. . . . .	98
5.9	RBM's hidden space features representation in an untrained model. . . . .	99
5.10	RBM's hidden space features representation in a trained model. . . . .	99
5.11	Learning curves with and without the CLBP-RBM preprocessing layer on the STL-10 dataset. Dashed lines denote results on training subset, solid lines on validation subset. . . . .	102
5.12	Evaluation of deep neural network performance depending on the number of training samples with and without the CLBP-RBM preprocessing on CIFAR-10 dataset. . . . .	104
5.13	Evaluation of deep neural network performance on all categories depending on the number of training samples with and without the CLBP-RBM preprocessing on Concrete Crack dataset. . . . .	105
5.14	Histograms for $P(\mathbf{v})$ taken from different datasets, cumulative difference denotes the sum of differences between a given bin to a reference bin from the first to the current bin. . . . .	108
5.15	Examples of noised images with different noise factors. . . . .	111
5.16	Validation learning curves for a denoising experiment, "GS" denotes number of Gibbs steps performed for reconstruction. . . . .	112
5.17	Examples of an image with (5.17b) and without (5.17a) added adversarial attack noise. . . . .	113
5.18	Decrease of accuracy on the testing dataset versus epsilon in adversarial attacks. . . . .	114
5.19	Difference in decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different Gauss noise parameters. . . . .	120
5.20	Difference in decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different ISO noise parameters. . . . .	120
5.21	Decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different distortion parameters, blue lines denote the difference in decrease of accuracy. . . . .	121

---

---

5.22	Comparison of the processing time and the number of parameters between the RBM+CNN and some commonly known deep neural architectures. . . . .	126
6.1	Hardware used for the mobile robot application. . . . .	128
6.2	General diagram of the experimental application. . . . .	129
6.3	Architecture of the robot's program. . . . .	131
6.4	Screenshot from the receiver application. Coordinates are shown in centimetres, "O" stands for $\theta$ . . . . .	132
6.5	Diagram of a mobile robot in its local coordinates. . . . .	133
6.6	JetBot's wheel velocity estimation. . . . .	135
6.7	Mobile robot error on performing one rotation with the use of CLBP-RBM and angle estimation. Negative value of the deviation denotes that the robot performed less than one rotation, positive value denotes more than one rotation. The error from visual feedback should be interpreted as random while the errors from angle estimation are systematic. . . . .	137
6.8	Training curves on validation dataset for the mobile robot object classification experiment. . . . .	140
6.9	Robot orientation task diagram, the robot should be positioned along the yellow line, $\beta$ is a localisation error. . . . .	142
6.10	Mobile robot angle deviation while positioning along an axis of a given object, 80 measurements for each category were conducted. . . . .	144
6.11	Histograms for $P(\mathbf{v})$ obtained from different datasets. . . . .	154
6.12	Training curves on the validation dataset for the mobile robot objects classification experiment with the use of different pre-trained RBMs. . . . .	155

---

## List of Tables

---

2.1	Example $3 \times 3$ patterns and their LBP8 output. . . . .	35
2.2	Average response times of different descriptors for a single key-point, times are relative to the slowest descriptor (SIFT). . . .	36
3.1	Possible scenarios of a single hidden unit energy value change during RBM state changes . . . . .	51
5.1	Example images from different datasets used for experiments.	84
5.2	Time in milliseconds of LBP8/CLBP response depending on the size of an input image with a squared shape. . . . .	86
5.3	Accuracy achieved by neural network with different number of convolutional layers with and without RBM preprocessing on MNIST dataset. . . . .	90
5.4	Example outputs of RBM feature extractor. The left image in each row is an input image, the images on the right are outputs from each of 24 hidden units of an RBM. . . . .	93
5.5	Accuracy improvements achieved by different types of network backbones with the CLBP-RBM preprocessing. . . . .	101
5.6	Accuracy reduction versus number of CNN layers reduction in "custom" backbone. . . . .	101
5.7	Evaluation of recognition pipeline generalisation ability metrics for each classified category on the testing dataset for concrete crack detection. . . . .	105
5.8	Chi-Squared distances (defined by (4.18)) between datasets measured with the use of an RBM's probabilities histograms, the diagonal results denote the distance between the reference set of images and another set from the same training dataset.	107
5.9	Reconstruction results. . . . .	110
5.10	Example patterns used for reconstruction, the original and with added random noise. . . . .	110

---

5.11	Example noised images with parameters used for the experiments ("cs" refers to colour shift in ISO noise, "i" refers to intensity). Images were generated with the use of Albumen-tations library [191] and the parameters of distortions refer to its settings. . . . .	117
5.12	Accuracy improvement of the CLBP-RBM preprocessing method for different types of noises. . . . .	119
5.13	Decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different Gauss noise param-eters, results are presented as "RBM/RAW". . . . .	119
5.14	Decrease of accuracy in a network with and without the CLBP-RBM preprocessing depending on different ISO noise param-eters, results are presented as "RBM/RAW". . . . .	119
5.15	Example images distorted with <i>Gaussian noise</i> and predic-tions from network with and without CLPB-RBM preprocess-ing. Green text denotes predictions with preprocessing, red without, predicted probability is given in parentheses. . . . .	122
5.16	The evaluation of a CD-based KNN matching procedure, re-sults are presented as "accuracy/mean Average Precision", the first row denotes "number of images for RBM training/num-ber of images for KNN training", the best result is highlighted with a bold font. . . . .	123
5.17	The evaluation of the accuracy of CD-RBM as an entry for DNN125	
5.18	The evaluation of network accuracy with RBM preprocessing and without it, "BD" refers to a binary descriptors layer. . . .	125
5.19	Validation accuracy depending on the size of the input image for an RBM with 10 hidden neurons and a CLBP descriptor as an input. . . . .	125
6.1	Objects chosen for robot classification system experiments. . .	139
6.2	Evaluation of recognition pipeline generalisation ability met-rics for each classified category on testing dataset. . . . .	139
6.3	Example screenshots from robot prediction, coordinates of the robot not displayed for the clarity of view. . . . .	141
6.4	Example recognition maps computed by mobile robot. Red dots denote camera, blue dots - AC, green dots - book. The bigger and brighter the dots the greater the certainty of oc-currence. . . . .	146

---

6.5	Example recognition maps computed by mobile robot. Red dots denote camera, blue dots - AC, green dots - book. The bigger and brighter the dots the greater the certainty of occurrence. . . . .	147
6.6	Example recognition maps computed by mobile robot. Red dots denote camera, blue dots - AC, green dots - book. The bigger and brighter the dots the greater the certainty of occurrence. . . . .	148
6.7	Example recognition maps computed by mobile robot. Blue dots denote closed door, green dots - open door. The bigger and brighter the dots the greater the certainty of occurrence. .	149
6.8	Example recognition maps computed by mobile robot. Blue dots denote closed door, green dots - open door. The bigger and brighter the dots the greater the certainty of occurrence. .	150
6.9	Example recognition maps computed by mobile robot. Blue dots denote closed door, green dots - open door. The bigger and brighter the dots the greater the certainty of occurrence. .	151
6.10	Example images from different datasets used for experiments.	153
6.11	Chi-Squared distances (defined in (4.18)) between the JetBot training datasets measured with the use of RBM's probabilities histograms, the diagonal results denote the distance between the reference set of images and another set from the same training dataset. . . . .	155



---

# ACRONYMS

---

<b>AI</b>	Aritificial Intelligence
<b>CD</b>	Contrastive Divergence
<b>CLBP</b>	Colour Local Binary Pattern
<b>CNN</b>	Convolutional Neural Network
<b>DBM</b>	Deep Boltzmann Machine
<b>DBN</b>	Deep Belief Network
<b>DL</b>	Deep Learning
<b>GPU</b>	Graphic Processing Unit
<b>LBP</b>	Local Binary Pattern
<b>LBP8</b>	8 bits Local Binary Pattern
<b>RBM</b>	Restricted Boltzmann Machine
<b>VAE</b>	Variational Autoencoder
<b>MLP</b>	Multi-Layer Perceptron
<b>MRF</b>	Markov Random Field
<b>SVM</b>	Support-Vector Machine





---

## NOTATION

---

$a, b, c, A, B, C$	-	scalars
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	-	vectors
$\mathbf{a} = [a_1 \ a_2 \ \dots \ a_n]$	-	n elements of $\mathbf{a}$ vector
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	-	matrixes, $\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \dots \ \mathbf{a}_n]$ denotes a matrix with $\mathbf{a}_i$ vectors being rows
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	-	tensors, $\mathbf{A} = [\mathbf{A}_1 \ \mathbf{A}_2 \ \dots \ \mathbf{A}_n]$ denotes a 3D tensor with $\mathbf{A}_i$ matrixes being channels
$\alpha, \beta, \gamma$	-	functions, or coefficients
$\mathbf{A}, \mathbf{B}, \mathbf{\Gamma}$	-	transformation with the use of set of functions $(\alpha, \beta, \gamma)$
$\mathbb{A}, \mathbb{B}, \mathbb{C}$	-	sets
$\mathbb{N}$	-	a natural numbers set
$\mathbb{N}_+$	-	a positive natural numbers set $(\{\mathbb{N} \setminus 0\})$
$\mathbb{R}$	-	a real numbers set
$\mathbb{A}^b$	-	a vector space with $b$ elements from $\mathbb{A}$ set
$\mathbb{A}^{n \times m}$	-	a space of matrices of size $n \times m$ with elements from $\mathbb{A}$ space
$\mathbb{A}^{d_1 \times d_2 \times \dots \times d_n}$	-	a space of n-dimensional tensors
$a_i$	-	an $i$ -th element from $\mathbf{a}$ vector
$a_{i,j}$	-	an element from $\mathbf{A}$ matrix, $i$ -th row, $j$ -th column
$a_{i,j,k}$	-	an element from 3D $\mathbf{A}$ tensor, $i$ -th row, $j$ -th column, $k$ -th channel
$\mathbf{A}_{i,:}$	-	an $i$ -th row of $\mathbf{A}$ matrix
$\mathbf{A}_{:,j}$	-	a $j$ -th column of $\mathbf{A}$ matrix
$\mathbf{A}_{:,:,c}$	-	a $c$ -th channel of a 3D $\mathbf{A}$ tensor
$\mathbf{A}_{i,j,:}$	-	all channels of a 3D $\mathbf{A}$ tensor at $i$ -th row, $j$ -th column
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	-	datasets

$X_{\sim \mathcal{U}}^{d_1 \times d_2 \times \dots \times d_n}$	-	randomly generated numbers with uniform distribution with respect to $d_1 \times d_2 \times \dots \times d_n$ dimensionality
$X_{\sim \mathcal{N}}^{d_1 \times d_2 \times \dots \times d_n}$	-	randomly generated numbers with normal distribution with respect to $d_1 \times d_2 \times \dots \times d_n$ dimensionality
$\{0, 1\}^{d_1 \times d_2 \times \dots \times d_n}$	-	a set of binary numbers with respect to $d_1 \times d_2 \times \dots \times d_n$ dimensionality
$\lfloor x \rfloor$	-	floor operation ( $\max\{m \in \mathbb{Z}   m \leq x\}$ )
$\lceil x \rceil$	-	round to a nearest integer ( $\lfloor x + \frac{1}{2} \rfloor$ )
$\mathbf{a}^t = (a_1^t, a_2^t, \dots, a_n^t)$	-	for dynamic systems: a state of $\mathbf{a}$ vector at time $t$
$\mathbb{E}_p[x]$	-	an expected value of $x$ over $p$ distribution
$\langle x \rangle_p$	-	an average expectation of $x$ over $p$ distribution: $\frac{1}{n} \sum_{i=1}^n (\mathbb{E}_p[x_n])$

## Abstract

*In recent years we have observed a dynamic growth of vision systems and artificial intelligence in science and many areas of industry. Due to computationally efficient resources, access to large quantities of data, and progress in science, it is possible to train very complex neural networks, that are capable of solving many image recognition problems, and whose effectiveness is comparable to human perception. However, the fact that solving complex classification problems often requires very large recognition systems has led to a focus on increasing the complexity of networks or enlarging their size, which in turn increases the complexity of recognition systems. Most of the currently used devices can handle those tasks, but it is not always possible to have access to efficient processing systems, which implies that large neural networks are not always applicable, especially in small devices, when price, size, and power consumption are the priorities. Another concern with complex neural architectures is that they require a large amount of training data, however accessing labelled training data is a frequent problem and may be expensive, time-consuming, and error-prone.*

*This dissertation presents a novel approach to visual data preprocessing that is focused on mitigating these difficulties. The proposed method introduces an additional stage of processing before the data is passed to a classification neural network. The classifier in this case may be a deep neural network which performs an additional high-level feature extraction, or a shallow network that utilises the previously extracted features directly. This stage is composed of two layers, the first transforms an RGB image data to its binary feature representation, the second is a small and fast neural network that utilises the binary data to obtain its most important features, which can be then used for classification. The output of the second layer is a real-valued tensor representing feature vectors for each image pixel, that can be directly passed as input of a regular neural network. An additional novelty is an improved local binary descriptor, which embeds colour information in its feature vector, instead of having only a code of the shape of a given part of an image. The main advantage of this approach is that the neural layer can be trained in a fully unsupervised manner, also its processing time is short in comparison to widely known neural feature transformers. The experiments performed in this dissertation demonstrate that adding these layers may be successfully applied to increase the overall accuracy of a neural network or decrease the size of a neural network without losing its quality.*

*Furthermore, the preprocessing proposed in this study tackles other common image processing problems. The experiments show its robustness in terms of input data denoising, and likewise introduces a special metrics that*

---

*can be used for comparing images, or measuring the similarity between image datasets to test if a given neural network can be used for transform learning.*

*The mentioned techniques tend to decrease the overall complexity of neural structures, hence their potential use is in embedded devices. Robotics is one of the areas that utilise these and vision in robots is an important part of their control systems and the size, cost, and power consumption of processing units are significant factors in design. Therefore, a part of this study focuses on an application of the proposed method in a mobile robot equipped with a single camera and a relatively low efficient processing unit. Experiments performed with this application demonstrate that the previously presented and analysed method can be successfully applied in a real device, thus the suitability of using the preprocessing layer in embedded systems is experimentally proved.*

### Streszczenie

*W ostatnich latach obserwujemy bardzo dynamiczny rozwój systemów wizyjnych i sztucznej inteligencji w różnych obszarach przemysłu, nauki i rozrywki. Wysoce wydajne zasoby sprzętowe, dostęp do dużej ilości danych oraz postęp badań w obszarze nowoczesnych technik identyfikacji i modelowania, przetwarzania danych i informatyki spowodowały, że możliwe jest stosowanie bardzo złożonych sieci neuronowych w charakterze uniwersalnych aproksymatorów. W efekcie, takie architektury mogą rozwiązywać wiele problemów klasyfikacji obrazów, a ich efektywność jest zbliżona do ludzkiej percepcji. W ogólności, rozwiązywanie złożonych problemów klasyfikacji często wymaga bardzo skomplikowanych systemów rozpoznawania. Badania w obszarze klasyfikacji obrazów skupiają się więc głównie na zwiększaniu złożoności sieci neuronowych lub bazują na wcześniej zdefiniowanych architekturach wymagających obliczeniowo. Większość obecnie używanych urządzeń może obsłużyć takie zadania, jednak dostęp do wydajnych jednostek obliczeniowych w niektórych przypadkach bywa ograniczony, przez co nie zawsze skomplikowane sieci neuronowe mogą być stosowane. Problem ten szczególnie dotyczy małych urządzeń gdzie ich rozmiar, cena i zużycie energii jest priorytetem. Dodatkowym problemem złożonych architektur jest to, że wymagają one bardzo dużych ilości oznaczonych danych trenujących, a dostęp do nich często jest trudny oraz może być kosztowny, czasochłonny i podatny na błędy.*

*Ta rozprawa doktorska przedstawia oryginalne podejście do wstępnego przetwarzania danych wizyjnych, którego celem jest zwiększenie efektywności rozpoznawania obrazów przy zachowaniu ograniczonego zapotrzebowania na moc obliczeniową. Zaproponowana metoda wprowadza dodatkową fazę przetwarzania bezpośrednio przed klasyfikującą siecią neuronową i proponuje etapy przetwarzania realizowane przez dwie warstwy. Pierwsza z nich przetwarza obraz RGB do jego reprezentacji w postaci wyekstrahowanych binarnych cech, druga jest niewielką siecią neuronową, która przetwarza binarne dane tak aby wydobyć z nich najważniejszych informacje i zależności między nimi, które można użyć dalej do klasyfikacji. Wyjście z drugiej warstwy jest tensorem wartości rzeczywistych, reprezentującą wektory cech dla każdego piksela. Tensor ten może być przekazany bezpośrednio do klasyfikującej sieci neuronowej. Klasyfikatorem w tym przypadku może być głęboka sieć neuronowa dodatkowo ekstrahująca cechy obrazu na wyższym poziomie abstrakcji lub płytka sieć neuronowa realizująca predykcję na wcześniej wydobytych cechach. Dodatkowym oryginalnym rozwiązaniem jest ulepszony lokalny deskryptor binarny, który w swoim wektorze cech koduje nie tylko kształt danej części obrazu, ale również informacje o jego kolorze.*

*Główną zaletą takiego rozwiązywania, jest to że warstwa neuronowa jest trenowana w całkowicie nienadzorowany sposób, a także czas przetwarzania*

---

*przez nią danych jest niski w porównaniu do innych znanych neuronowych detektorów cech. Wyniki badań eksperymentalnych prowadzonych w ramach tej pracy wskazują, że dodanie proponowanych warstw przetwarzania może być z powodzeniem zastosowane w celu zwiększania jakości rozpoznawania przez sieci neuronowe, a także do zmniejszenia ich rozmiarów bez utraty jakości.*

*Zaproponowana metoda może być również użyta do rozwiązania innych powszechnie znanych problemów przetwarzania obrazu. Eksperymenty wskazują na celowość jej stosowania w przypadku odszumiania danych wejściowych. Wprowadza ona także specjalną metrykę, która może być zastosowana do porównywania obrazów, lub szacowania podobieństwa zestawów obrazów w celu testowania czy sieć neuronowa o danych parametrach może być użyta do rozwiązania innego problemu klasyfikacji bez konieczności dodatkowej optymalizacji wag.*

*Jednym z celów badań prowadzonych w ramach tej rozprawy jest dążenie do zmniejszenia ogólnej złożoności struktur neuronowych, przez co ich potencjalny obszar zastosowań to systemy wbudowane. Robotyka jest jedną z dziedzin która na nich bazuje, ponieważ wizja w systemach sterowania stanowi często ich istotną część. Ponadto czas, rozmiar i koszt jednostek obliczeniowych jest znaczącym czynnikiem w projektowaniu urządzeń przemysłowych z integrowanymi systemami rozpoznawania obrazu. W związku z tym, część pracy skupia się na aplikacji zaproponowanej metody w podsystemie percepcji kołowego robota mobilnego wyposażonego w jedną kamerę i relatywnie nisko wydajną jednostkę obliczeniową. Wykonane testy wskazują, że przedstawiony i przeanalizowany schemat przetwarzania danych wizyjnych może być skutecznie zastosowany w rzeczywistym urządzeniu. Pozwoliło to na eksperymentalne potwierdzenie słuszności użycia technik opracowanych w niniejszej rozprawie dla pewnej klasy systemów wbudowanych.*

---

# INTRODUCTION

---

## 1.1 MACHINE VISION

---

The way machines may see and interpret the world has fascinated scientists for decades. Starting in the early 1960s, the first attempts to investigate how images can be processed by a computer and also how the brain processes visual signals were made. The optimism related to the possibilities of AI in vision was then high in many areas including robotics, but the limited resources and computing power at the time disappointed those expectations quickly, and the progress of machine vision in real applications was hindered. Nevertheless, the many forms of research done many years ago have created a very solid mathematical background for the robust image processing techniques we know today. Now computer vision is highly connected to artificial intelligence, but the first challenges were mostly related to digital signal processing. A single image is now often represented by a three-dimensional matrix (respectively height  $\times$  width  $\times$  channels), which implies the possibility of applying any linear algebra or other discrete transformation to an image to amplify or suppress desired information. That is, computer vision is nothing else than the adoption of the commonly known mathematical methods to an image, thus image processing becomes de facto a transformation of an input matrix to its desirable form.

The significant progress in electronic technology has resulted in cameras being almost everywhere in our surrounding environment, so the usability of computer vision is almost unlimited. Even devices considered today as low-performance may perform some basic image processing to obtain relevant visual information. As a consequence, many of the methods used in vision systems can be easily applicable in robotics or other automatic systems with visual control feedback to support their perception. As computer vision generally focuses on interpreting the world based on images, machines may similarly perceive their surrounding. There is a variety of tasks in robotics where image processing may help, and its usability depends on particular applica-

tion requirements. Thus, it is important to mention that focusing on computer vision as a general tool may result in more efficient robotics systems. Nevertheless, the most important challenges are classification, segmentation, and object detection. A correctly recognised scene leads to enriching the robot perceiving loops in much crucial information about the environment it is located in, hence it can execute more precise actions. Visual perception can provide essential information in context of robot localisation, decision making and modelling of the environment. Therefore, computer vision in robotics has been the subject of research for many years [1, 2, 3, 4, 5, 6, 7].

Many computer vision problems are close to being solved, which means that their processing ability is at the human perception level [8]. However, this field is still a considerable area of interest for science, because there are many other difficulties. Mostly, as this dissertation focuses on, they are related to data availability and the computing power required for them which will be described in more detail in the following sections. Firstly, however, a general concept of neural networks will be introduced as they are the fundamental techniques to address most computer vision challenges. Secondly the main problems consider in this thesis will be discussed.

## 1.2 NEURAL NETWORKS

---

### 1.2.1 Basic concepts of neural networks

---

Every artificial neural network is composed of its elementary units - artificial neurons. They were introduced in 1943 by McCulloch and Pitts in [9], where an untrainable mathematical model representing the functionality of a biological neuron was described. Then the model was modified, by introducing training rules and minor modifications of the processing formula, but the basic idea remained unchanged and the artificial neuron that is known and widely used today is schematically shown in Figure 1.1.



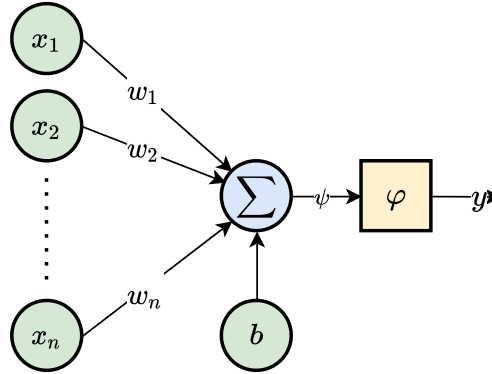


Figure 1.1: Diagram of an artificial neuron.

Formally, one neuron processes  $n$  scalar inputs (or a vector  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^\top$ ), and generates only one output, which can be described as follows:

$$y = \varphi\left(b + \underbrace{\sum_{i=1}^n w_i x_i}_{\triangleq \psi}\right), \quad (1.1)$$

where  $\varphi$  is a non-linear activation function,  $w_i$  is the weight of the neuron related to the input  $x_i$ ,  $b$  is a bias,  $\psi$  is a linear combination of inputs and weights with added bias. Therefore all the parameters in a single neuron form a vector -  $\mathbf{w}$  of size  $n + 1$  -  $\mathbf{w} = [b \ w_1 \ w_2 \ \dots \ w_n]^\top$ . The usability of a single neuron is very limited, but many neurons may form a more complex model by the use of outputs of some neurons as inputs for others. There are many topologies of such connections, but it is sufficient to limit the considerations to Multi-Layer Perceptron (MLP) to present the principles of neural networks because it is a standard in the AI nowadays and some other architectures, even recurrent ones can be reinterpreted as similar to an MLP. This is also an important model for this study, thus it has been chosen as a representative example of neural network architecture.

An MLP is a deterministic model composed of some number of layers -  $m$ , where each layer composes of a number of neurons -  $[n_1 \ n_2 \ \dots \ n_m]^\top$ , an input signal propagates from the input to the output layer, and the neurons are fully connected (every output from the previous layer is connected to every input in next layer). The layers between input and output are referred to as *hidden layers*, schematically this is shown in Figure 1.2,  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^\top$  - denotes an input signal,  $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_n]^\top$  is an output signal,  $\mathbf{h}_i = [h_1 \ h_2 \ \dots \ h_n]^\top$  are intermediate hidden signals. For further discussion -  $\mathbf{h}$  is generalised to all units in the network, and  $\mathbf{h}_1$  is denoted also as input, and  $\mathbf{h}_m$  as output,

so  $\mathbf{h}_1 \triangleq \mathbf{x}$  and  $\mathbf{h}_m \triangleq \mathbf{y}$ . For the sake of clarity, it is worth mentioning that the terminology "MLP" may be ambiguous because one can understand that as a multilayer network composed of "perceptrons" [10], but a perceptron output is binary because applying a Heaviside function as  $\varphi$ . However, this is not a feature of an MLP, which can utilise any activation function.

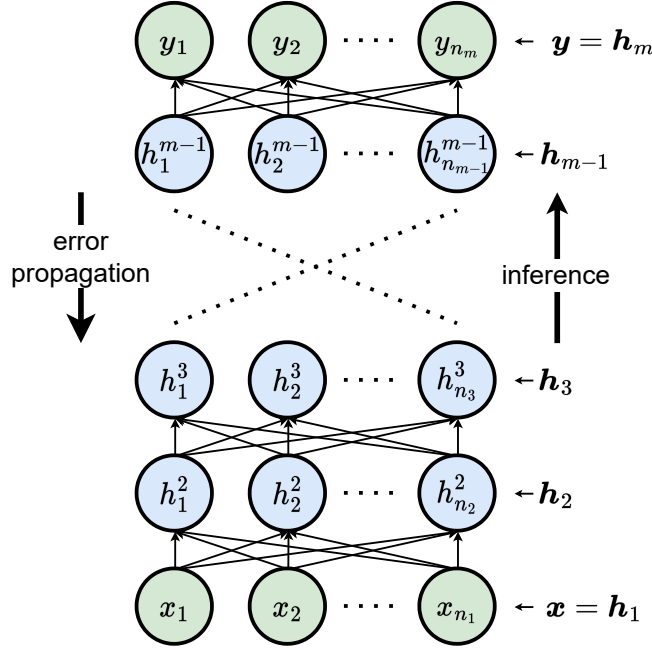


Figure 1.2: Diagram of a Multi-Layer Perceptron.

Now, one can define the number of parameters of a single  $i$ -th layer in an MLP:

$$N_i = \underbrace{n_i}_{\text{no. units in layer}} \times \underbrace{(n_{i-1} + 1)}_{\text{no. units in previous layer}} . \quad (1.2)$$

The number of all parameters in the network is then equal to  $N = \sum_{i=2}^m N_i$ . This parameter may be very large, especially for high dimensional input and many hidden layers.

The non-linearity between the layers is important because otherwise, the hidden layers would not be necessary. To exemplify, consider two linear layers of an MLP, as shown in Figure 1.3.

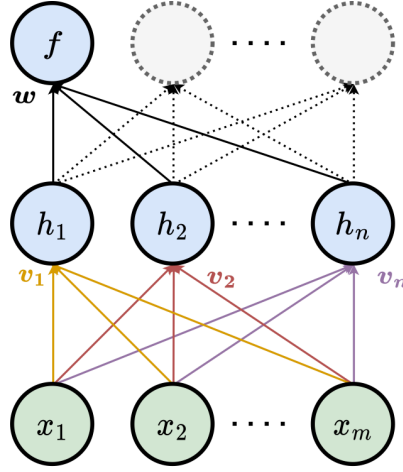


Figure 1.3: Diagram of two layers of a Multi-Layer Perceptron.

The first hidden layer processes an input  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_m]^\top$  composed of  $n$  neurons, where each of them has its own weights  $\mathbf{v}_i = [v_{i,1} \ v_{i,2} \ \dots \ v_{i,m}]^\top$  and performs the following linear transformation:

$$h_i = \mathbf{v}_i^\top \mathbf{x}, \quad (1.3)$$

hence the hidden layer vector has the following form:

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^\top \mathbf{x} \\ \mathbf{v}_2^\top \mathbf{x} \\ \vdots \\ \mathbf{v}_n^\top \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_n^\top \end{bmatrix} \mathbf{x}. \quad (1.4)$$

Consider the transformation function of the linear unit from the next layer, with weights vector  $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_n]$ :

$$f = \mathbf{w}^\top \mathbf{h} = \mathbf{w}^\top \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_n^\top \end{bmatrix} \mathbf{x} = \tilde{\mathbf{w}}^\top \mathbf{x}, \quad (1.5)$$

As a result, all the weights in the analysed layers could be replaced with  $\tilde{\mathbf{w}} = [\tilde{w}_1 \ \tilde{w}_2 \ \dots \ \tilde{w}_m]$  being a linear combination of  $\mathbf{w}$  and all  $\mathbf{v}_i$ . For  $h_i$  being a neuron with a non-linear activation function this is not possible and all the layers process the signals with more abstract meaning. With the use of non-linearity, an MLP may solve many complex problems, therefore those linearly separable are rather trivial.

**Example 1** (XOR classification problem). The most commonly known example of a simple non-linear case is an XOR problem [11] - a linear unit cannot perform the function determined by the following set of data:

$$\mathcal{S} = (\mathcal{X} \in \{0, 1\}^2, \hat{\mathcal{Y}} \in \{0, 1\}) = \{([0 \ 0], 0), ([0 \ 1], 1), ([1 \ 0], 1), ([1 \ 1], 0)\},$$

where  $\mathcal{S}$  is a training dataset split into  $\mathcal{X}$  - a set of input signals and  $\hat{\mathcal{Y}}$  - a set of desired outputs.

**Example 2** (Circle pattern classification problem). Another good example of how hidden layers increase the level of abstraction (inspired by [11]) is a classification of a circle pattern in Cartesian coordinates. It is not possible to separate it with a single neuron. However, intuitively we can transform the coordinates to be polar, and then the radius determines to which class a given example belongs. In the area of neural networks, we can train an MLP to recognise it. To make it more concrete, an MLP with two hidden layers ( $\mathbf{g}$  and  $\mathbf{h}$ ) was experimentally trained. The results are shown in Figure 1.4, which illustrates how a neural network processes the data with non-linearities to be separable linearly in the last layer.

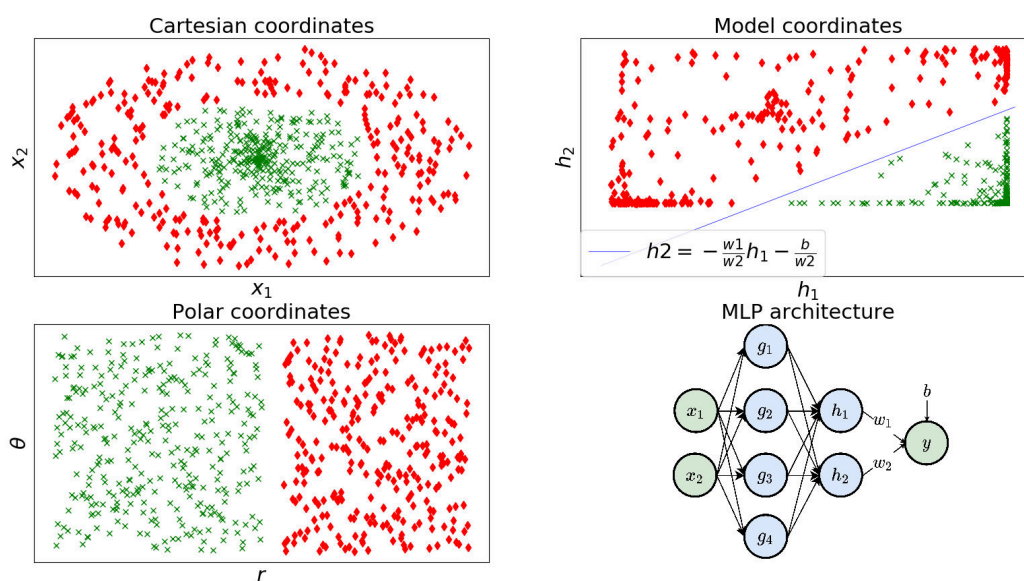


Figure 1.4: Example of a non-linear classification problem with its representation in other spaces (polar and given by the MLP), the classes are marked by two colours, the blue line in the model coordinates shows how the last neuron separates the  $(h_1, h_2)$  surface. The second hidden layer of the MLP is not necessary but allows the projection of the hidden space onto 2D surface.

Currently, the most common non-linearity is the Rectified Linear Unit (ReLU) [12] ( $\varphi_{ReLU}(x) = \max(0, x)$ ), or some of its variations [13], but earlier it was sigmoid or hyperbolic tangent function. The reason why these have been superseded by ReLU is mostly related to the vanishing gradient problem [14], which means that during the training phase gradients become very low in the first layers after propagation from the top layers, moreover ReLU can be computed much faster.

MLP is a deterministic model and signals in a single layer propagate synchronously, each layer performs a non-linear transformation of an input that can be defined by a vector:

$$\begin{aligned} \mathbf{h}_i &= [\varphi(\psi_1) \ \varphi(\psi_2) \ \dots \ \varphi(\psi_{n_i})]^\top \\ &= [\varphi(\mathbf{h}_{i-1}, \boldsymbol{\theta}_{i_1}) \ \varphi(\mathbf{h}_{i-1}, \boldsymbol{\theta}_{i_2}) \ \dots \ \varphi(\mathbf{h}_{i-1}, \boldsymbol{\theta}_{i_n})]^\top \\ &= \mathbf{f}_i(\boldsymbol{\theta}_i, \mathbf{h}_{i-1}), \end{aligned} \quad (1.6)$$

where  $\boldsymbol{\theta}_i$  denotes all the parameters in an  $i$ -th layer,  $\boldsymbol{\theta}_{i_j}$  is a vector of parameters of a  $j$ -th neuron in an  $i$ -th layer,  $\psi_j$  is a total input to a given neuron  $j$  as described in (1.1) and  $\mathbf{f}_i(\cdot)$  is a transformation function of an  $i$ -th layer. Finally, one can define the entire MLP architecture as a non-linear transformation of an input defined by the following equation.

$$\mathbf{y} = f_m(\boldsymbol{\theta}_m, f_{m-1}(\boldsymbol{\theta}_{m-1}, f_{m-2}(\dots (f_2(\boldsymbol{\theta}_2, \mathbf{x})))))) = f(\boldsymbol{\theta}, \mathbf{x}), \quad (1.7)$$

where  $\boldsymbol{\theta}$  denotes all the parameters in the network. The major challenge in the AI field is obviously to adjust all the weights to solve a given problem. For given sets of signals  $\mathcal{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_n]^\top$ ,  $\widehat{\mathcal{Y}} = [\widehat{\mathbf{y}}_1 \ \widehat{\mathbf{y}}_2 \ \dots \ \widehat{\mathbf{y}}_n]^\top$ , we want  $\mathbf{y} = f(\boldsymbol{\theta}, \mathbf{x})$  (model output) to follow given  $\widehat{\mathbf{y}}$  which is the desired output. This can be considered as an optimisation problem for determining  $\boldsymbol{\theta}$  for any given set of input and output signals. Having defined the number of parameters and knowing the high dimensionality of the  $f$  function, one can conclude that determining  $\boldsymbol{\theta}$  may be not possible analytically in many cases, thus it has to be computed algorithmically. The optimisation algorithm defines a loss function  $e(\widehat{\mathbf{y}}, \mathbf{y})$  describing a distance between  $\widehat{\mathbf{y}}$  and  $\mathbf{y}$ , and then iteratively changes the parameters minimising the  $e$  function until satisfying some initially defined conditions (for example the value of  $e$  less than some threshold, or the number of iterations bigger than some threshold). The general formula is defined as follows:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \underbrace{\Delta \boldsymbol{\theta}^t}_{\text{depends on } e(\widehat{\mathbf{y}}, \mathbf{y})}. \quad (1.8)$$

The process of adjusting  $\boldsymbol{\theta}$  may be performed with the use of a gradient descent method, which in short, for given loss value -  $e$  computed in one step,

takes advantage of a gradient  $\frac{\partial e}{\partial \theta}$ , and tends to minimise its value. Currently, it is often done with a backpropagation algorithm, popularised by Hinton, however discovered independently by several groups of scientists [15, 16, 17, 18]. This technique propagates the error backward from the output layer to the first hidden layer, thus with the use of chain rule one can define a gradient for every single weight in the model -  $\frac{\partial e}{\partial \theta_{ijk}}$  [18] and the formula of updating is given as:

$$\Delta \theta_{ijk}^t = -\eta \frac{\partial e^t}{\partial \theta_{ijk}^t}, \quad (1.9)$$

where  $i, j, k$  refer to an  $i$ -th layer,  $j$ -th neuron, and  $k$ -th weight,  $\eta$  is a small positive scalar value that scales the update term. The entire procedure is called *learning* or *training*, and if  $\hat{\mathcal{Y}}$  is prior given (by human labour) it is called *supervised learning*, otherwise *unsupervised learning* ( $\hat{\mathcal{Y}}$  - is automatically inferred).

In general, neural networks may solve different types of problems; however, this study focuses mostly on classification problems. In this case,  $\mathbf{y}$  from (1.7) takes a form of vector  $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_n]$ , where each  $y_i$  denotes how likely a given input belongs to each from  $n$  categories. Usually, the output is transformed to a more useful form -  $\tilde{\mathbf{y}}$ , such as  $\forall_i \tilde{y}_i \in [0; 1]$  and  $\sum_i \tilde{y}_i = 1$  which is a result of applying a softmax function [11] after the last layer in a network:  $\forall_i \tilde{y}_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$ .

### 1.2.2 Neural networks in vision

Visual recognition methods changed significantly in 2012, when Krizhevsky, Hinton and Sutskever revealed a deep neural network - AlexNet [19] that outperformed all previously known models in the *ImageNet Large Scale Visual Recognition Challenge* [20]. It was a very complex task for image classification systems, because of the large amount of data being classified (more than 1.2 million) and number of classes (1000). AlexNet achieved 15.3% top-5 error, which was more than 10 percentage points better than its competitors. It utilised Convolutional Neural Network (CNN) as a base architecture. Convolutional layers are explained in more detail in the next chapter, but in short, a CNN-based classification is composed of many convolutional layers used as a feature detector and several fully connected layers (MLP) used as a feature classifier which is schematically presented in Figure 1.5.

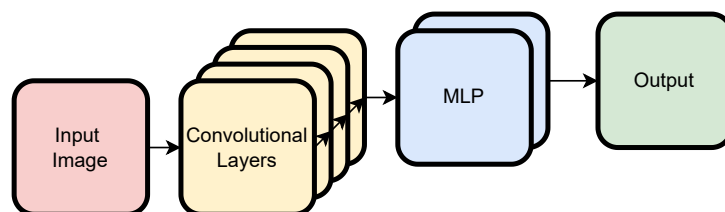


Figure 1.5: Simplified diagram of a CNN neural network.

CNNs are still used in many neural models for image processing, and they are usually very complex. One entire field in science that among others focuses on these is named *deep learning* (DL); this term de facto has no strict definition, but broadly speaking, its modern interpretation refers to artificial neural networks composed of many hidden layers. The main concepts of DL are given by LeCun, Bengio and Hinton in [21] - a highly influential article describing how these models work, and why their recognition ability is substantial.

In fact, CNNs were known before 2012, and the article that presented this type of network for handwritten digit recognition from 1989 published by LeCun [23] is considered a precursor of CNNs. These networks were also applied in other vision tasks at the time, for example for face recognition [24]. However, AlexNet was a milestone in the entire area of deep learning. Not only because of the first use of CNN in large scale datasets but also of some novelties, among others the heavy use of a graphic processing unit (GPU) in training a large deep neural network. Although it was not the first attempt to employ GPU for CNN [25, 26], the previous results were obtained for much smaller models, AlexNet also used two GPUs for distributed training. In the next decade, CNNs and training them on GPUs became a standard in artificial intelligence, and parts of AlexNet architecture have been successfully applied in various machine vision tasks such as object detection [27], region segmentation [28], human pose estimation [29], video classification [30], object tracking [31], and image upscaling [32]. The main reason that it was achievable is that CNN and its particular implementation - AlexNet is a robust image feature transformer that can be used in a variety of scenarios; however, the classification results on the ImageNet dataset have been being consequently improved, among others the most important models that outperformed AlexNet were GoogleNet [33], InceptionV3 [34], VGG [35], ResNet[36], DenseNet [37], EfficientNet [38]. Although they differ in terms of their architectures, they share the same feature extraction idea, which is a CNN. This resulted in artificial intelligence becoming an inseparable part of modern society, and many major companies like Facebook, Google, Mi-

crosoft, IBM, Yahoo! Twitter, and Adobe invest significant resources for research and developments in this area of industry [21]. Furthermore as DL has some special hardware requirements we can observe many changes in chip development to support the computations needed for it. Nvidia, Intel, Apple, and Qualcomm develop their processors to take deep learning needs into account. Hence, we can say, we are witnessing a revolution in industry and economy which is a result of research in artificial intelligence. Moreover, DL has also a big impact on a variety of fields of science, since its modalities are miscellaneous, it is applicable to different types of scientific problems and as explained in [22] there is considerable growth in the adoption of DL techniques as a research tool.

In fact, a number of recent works prove that convolutional layers are not necessary to achieve good performance [39, 40], but CNN-based networks are still the most commonly used models in computer vision systems and this study focuses mainly on these.

### 1.2.3 Neural networks in robotics

---

In today's world, robotics plays an important role in almost every part of our life. Manipulators, mobile robots, and autonomous vehicles support us in industry, medicine, the military, and some of our daily duties. We notice that progress in robotics tends to result in robots behaving more like humans or at least having some similar properties in terms of recognition and interaction with an environment. This implies that neural networks are highly important for robots, notably in their vision which this dissertation relates; however, AI has many other use cases for robots like perception, decision-making, high-level control, or updating of controller parameters. Nevertheless, AI in robotics does not solve any control problems itself, it should be regarded mostly as a tool that may assist in better environment understanding and supporting their control loops.

For a brief historical background, it is worth mentioning that AI in general was applied to robotics problems even before the 1970s, the first significant achievement was Shakey Robot [41] which was a mobile robot being able to perceive the environment, reason, and solve some relatively complex task using the first AI algorithms, however without utilising neural networks. The implementation of neural networks for control is known since the 1980s when in [42] described how AI may support in control of robot manipulator. Later some researchers in the 1990s described similar and other use cases [43, 44, 45]. An important study was presented in 1989 [46] wherein the authors proposed the use of a neural network for mobile vehicle image



recognition. The network usability then was obviously very limited when compared to more recent achievements; however, still very impressive taking the computing power and knowledge of AI at that time into account. Moreover, it introduced the idea of neural vision processing in robotics which is the field that this study focuses on.

The significant progress in artificial intelligence and computing power of robots processing units has resulted in fewer limitations in the usability of neural networks, and one can find many studies showing example applications of neural models in almost any field related to robotics, such as robotic grasping [47, 48], path planning and navigation for mobile robots [49, 50], sensor data processing [51, 52], obstacle detection and avoidance [53, 54], road detection [55, 56]. As one may imagine there is a variety of others since AI performs best in many other areas that robots use, for example, scene classification, object localisation, audio recognition, and processing, etc., so any particular application depends only on the task requirements. Thus one can conclude that as robotics relies on the theoretical principles provided by the research on AI, there is a need to work on strict neural network problems to provide more efficient tools for robotics.

The property that distinguishes robotics from others fields of industry is that it the available resources are often limited and the robotic systems have to meet real-time capability requirements. The limitations are as a result of the size of robots, their power consumption requirements, and the price in some cases. Therefore, in robotics the size and speed of neural networks become more significant, and it is highly important to make them as fast as possible to perform any given task. Another difficulty is data availability as previously mentioned, but in robotics this problem arises more frequently because as the robots often work in restricted environments. Thus, the data they process is limited, and may also be problematic to be known prior. It implies that supervised learning in this case may be difficult and expensive because of data labelling. Hence, the goal, in this case, is to make use of unlabelled data provided by robot and limit the amount of labelled data needed for efficient training.

Considering this scenario, one can ask if a robot that has been adapted to work in a particular environment can be moved to another one and perform the recognition task with a similar accuracy. This is a complex problem as standard neural networks do not provide a metrics that would inform if transfer learning is possible, which means whether or not the transferred part of a network is a good feature extractor for a different recognition problem.

This study addresses these difficulties. Foremost, it presents an original preprocessing technique that permits the limitation of the size of a network, enabling better recognition efficiency when the unlabelled data is available,

but labelled data is limited, and finally, it provides a metrics to compare the distribution of a training dataset to new data, without retaining the original data.

### 1.2.4 Deep neural networks complexity

In general, a major concern related to CNN-based models and large neural networks is that they are composed of many parameters which increases their possibilities to accurately approximate even highly nonlinear functions, but also makes them computationally demanding. The comparison between size and accuracy of some representative CNN models is shown in Figure 1.6, but before an analysis of the results, it is important to mention that some of the models outperform human recognition ability, which is at level of 95% [57]. It indicates the importance of applying DL techniques in image processing tasks.

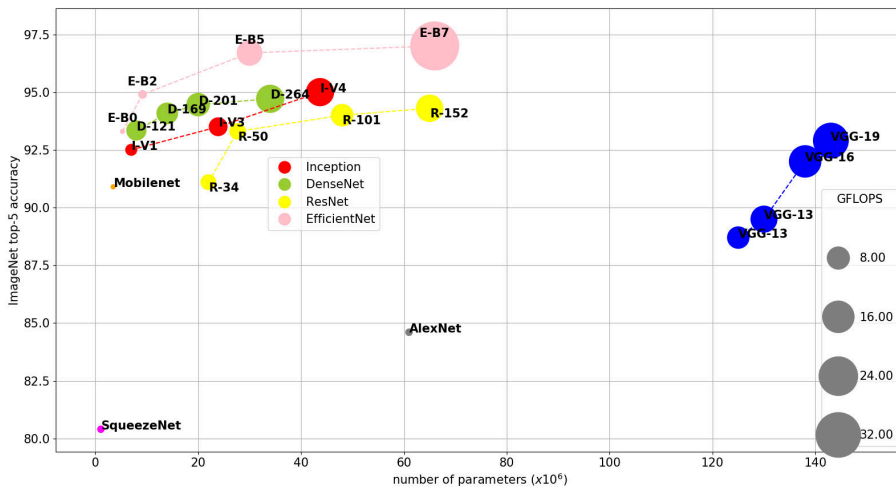


Figure 1.6: ImageNet top-5 accuracy of some common CNN architectures versus their sizes [19, 33, 34, 35, 36, 37, 38, 58, 59, 60].

The first observation is that the number of parameters and floating point operations is very large for every model, so all the architectures that achieve good performance on the ImageNet dataset are very resource-consuming. This is obviously something that can be expected because complex tasks require many parameters in the model to be solved. However, this leads to a problem with computing power needed for training and inference. Some models like SqueezeNet [58] or Mobilenet [59] are designed to be relatively small and still achieve a good accuracy, but their overall performance is worse

than their bigger competitors. EfficientNet model [38] tackles the scalability task, and is also one of the best models currently available, but still the dependency between size and accuracy is clear as it is for other models shown in this comparison. Therefore, a versatile method that could increase the performance of a particular model without significant increase of its size would be useful for many reasons, especially for its application in systems with limited resources. Another concern with large neural models is that they need large quantities of training data to achieve a good performance [61] which may be problematic, particularly when the model has to be learnt from scratch and no transfer learning [62] can be applied. The models mentioned earlier were trained on a large and precisely labelled images set - the ImageNet, but for many practical scenarios, the process of preparing a training set has to be performed on other images. The data is usually available or can be easily gathered, the problem is with labelling it. For supervised learning which is used to train the above-mentioned neural network, the data has to be categorised by humans, which is time-consuming, error-prone and relatively expensive. This is why there is a need to make use of unlabelled data.

### 1.3 RESEARCH PROBLEMS AND PROPOSED SOLUTIONS

---

As mentioned previously, neural networks that achieve state-of-the-art results in many commonly known problems are composed of many parameters and require many floating point operations to compute the output. This can be an advantage, as having a large number of training samples and computationally efficient processing units causes that they are applicable in many scenarios. However, these conditions are often not met, especially in systems with limited resources. There are many such systems, especially embedded platforms, that have restricted computing efficiency because the size, price, and power consumption are significant factors when designing the systems. Robotics and particularly mobile robots are the parts of the industry that utilise this type of systems, therefore there is a considerable need to reduce the resources needed to solve a given classification problem.

The second important concern in image recognition in robotics is data availability. In general, mobile robots are able to gather image data, but they cannot make use of them without labelling by human, thus the research on unsupervised learning methods that do not require lots of computing power is highly important.

Based on the aforementioned problem, the research presented in this dis-

sertation focus on solving two major problems:

- reducing the number of parameters in an image classification network while satisfying the accuracy requirements,
- using unlabelled data to increase the accuracy of CNN networks when labelled data is limited.

As will be shown in the following chapters, these two problems are related, since there is a possibility to preprocess the data in an unsupervised way to have more abstract features in the input. So both of the research problems can be solved by the same approach.

The author of this study proposed a novel structure of image processing that introduces an additional stage of processing before the CNN, as presented in Figure 1.7.

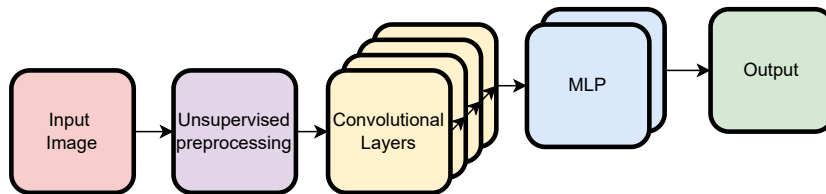


Figure 1.7: Simplified diagram of a CNN neural network with unsupervised preprocessing.

The preprocessing composes of two layers, the first is a transformation to a binary space that describes some basic features of an image, and the second is a neural network that processes these descriptors to obtain the most important features and transforms them into a more abstract space. Additionally, the neural network is a recurrent model that can learn the probability distribution an input data. Taking this into consideration, it is possible to define two additional important abilities that solve other image processing problems:

- reconstruction of input data if it is corrupted or affected by noise,
- comparison of given images dataset to the one that has been used for training - a measure of similarity of visual data.

Another problem, related directly to binary image descriptors will be raised in the following chapters. In short, these are relatively good feature extractors but there are not many effective methods that would process them for classification purposes especially when the number of descriptors from an

image may vary. This dissertation shows that binary descriptors may be aggregated for classification purposes with the use of the same recurrent neural network; however, the rest of the pipeline changes as the output of aggregation is more an expanded feature vector that represents the entire image rather than the image processed to another space.

Another unique achievement in this study is a novel binary feature descriptor that enhances the commonly known approach by additional bits that represent the colour of a given part of an image.

The implementation of the novel descriptor is prepared by the author as well as an implementation of the previously mentioned recurrent neural network.

Finally it is possible to formulate the following thesis statements:

- it is possible to increase generalisation ability of a neural network with the use of the novel unsupervised preprocessing method based on binary descriptors and a recurrent neural network,
- this preprocessing may be applied to reduce the complexity of a classification neural network without a significant decrease in accuracy, thus minimising required computing resources,
- applying the proposed preprocessing may result in less image-distortion sensitivity of a classification neural network,
- this preprocessing may provide an additional metrics allowing to measure a distance between image datasets.

It is not possible to verify the above-mentioned statements analytically as the structures of neural networks are too complex, hence commonly known statistical methods will be employed to show the potential usability of the proposed method and to compare with models not utilising it.

## 1.4 DISSERTATION SCOPE AND STRUCTURE

---

The dissertation focuses on neural networks in machine vision and their general usability in computer science. However, the main areas of utilisation of the proposed solutions are systems where the computing power or data availability is limited. Chapter 1 has presented a general overview of current knowledge in this field and also has presented some basic concepts of neural networks for a better understanding of the models being described in the next part.

Chapter 2 explains the most commonly known and used image feature extractors, which are the essential part of the proposed approach in this study. This chapter describes the trainable neural extractors and also feature descriptors that are formally transformations with prior given parameters. An important section is 2.5, which presents a novel approach to include colour information to a feature shape descriptor. Chapter 3 is devoted to the Restricted Boltzmann Machine as it is a base for all the considerations in this research. Chapter 3 presents the general concepts of this model, including the training and use cases for more complex architectures.

At the beginning of Chapter 4, there is a brief explanation of how the novel feature descriptor and the Restricted Boltzmann Machine were implemented, to achieve the best performance and flexibility for experiments. Chapter 4 presents also the detailed method of how RBM and binary descriptors can be used in terms of image processing. It describes all the techniques from a formal point of view in order to implement them in any programming tool. Additionally, it presents all the parameters that have to be set/adjusted in order to achieve the best performance. Chapter 5 is a presentation of the results of the tests that have been performed during the research. It shows experimental proof of the suitability of applying the methods described in Chapter 4. Chapter 6 describes a practical application of the previously presented methods. As the study focuses on some problems that highly relate to robotics, the application is implemented to be an example of mobile robot perception. Chapter 6 shows the details of image processing pipelines implementation as well as the results achieved when compared to an architecture without the proposed preprocessing. Chapter 7 contains the conclusions from the research and experiments.

---

## IMAGE FEATURE ENGINEERING

---

Image classification in its general form composes of two stages. The first is an extraction of visual features, the second is a classification of them, a diagram is presented in Figure 2.1.

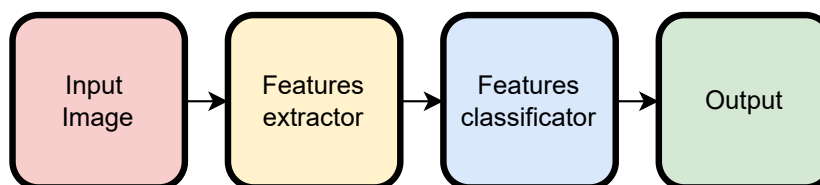


Figure 2.1: Simplified diagram of a typical image classification pipeline.

There are many ways to obtain the most important information from images. The feature extraction is a transformation of their raw pixel representation to a space that can be separated by a classifier. Raw pixel data is easily recognisable by humans because of how the brain works, but for the conventional machine learning algorithms like MLP or Supported Vector Machines (SVM) [63] the raw data is too complex and often its dimensionality is too big. In other words, the feature extraction is amplifying only the information from an image that are needed for classification and suppressing the irrelevant ones. For decades, feature engineering was a very complex task because it required careful engineering and excellent knowledge of the data domain [21]. It was usually composed of the detection of some basic features like edges, blobs, corners, lines, etc., because the high-level features like shapes or dependencies between the lower level features were hard or impossible to detect. Nevertheless, deep learning changed significantly the dominant approach to image classification because feature extraction has become a trainable process. That is a fundamental reason why DL models are important tools as they can learn how to process an image to extract the features. However, as mentioned before it requires a lot of training data and computing power.

---

## 2.1 CONVOLUTIONAL LAYERS

---

The convolutional layer is an essential feature extractor in most commonly used DL architectures. The way it works is inspired by the receptive fields in animal brain [64]. It can be conveniently described in a mathematical language using the convolution/split operator denoted by  $*$  symbol, in the continuous domain it can be defined by:

$$(h * g)(t) \triangleq \int_{-\infty}^{\infty} h(\tau)g(t - \tau)d\tau, \quad (2.1)$$

In the discrete domain the split can be represented by a sum operation. In addition, assuming that  $\mathbf{h} = [h_1 \ h_2 \ \dots \ h_n]^\top$ ,  $\mathbf{g} = [g_1 \ g_2 \ \dots \ g_n]^\top \in \mathbb{R}^n$  are vectors one can consider the following:

$$(\mathbf{h} * \mathbf{g})_n \triangleq \sum_{i=1}^m h_i g_{n-i}. \quad (2.2)$$

Formula (2.2) is the core of CNN layers,  $\mathbf{h}$  in this case is called *filter* or *kernel*, and  $\mathbf{g}$  is an input image. A single CNN layer is built of many filters, where each filter is specifically parametrised to be reactive on some particular pattern. Taking into account the two dimensional image nature, a convolutional filter is a matrix of size  $m \times n$ :

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix}, \quad (2.3)$$

where the  $w_{i,j}$  is a single weight in a filter. The feature type that is being amplified by the particular filter depends on the dependencies between the weights. For an abstract example, a  $3 \times 3$  kernel could learn to be a vertical edge detector and have a form similar to a Sobel filter [65]:

$$\mathbf{W} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \quad (2.4)$$

From a mathematical point of view, an output of a single convolution filter in a single place of an input image, is a sum of bias  $b$  and element-wise product of a filter weights and processed part of an image followed by an activation function.

$$o = \phi(\mathbf{I}, \mathbf{W}, b) = \varphi \left( \left( \sum_{j=1}^m \sum_{k=1}^n w_{j,k} \cdot i_{j,k} \right) + b \right), \phi : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}, \quad (2.5)$$



where  $\varphi$  is a non-linear activation function,  $w_{j,k}$  denotes filter weights,  $b$  is a filter bias,  $i_{j,k}$  denotes an element from input matrix -  $\mathbf{I}$ . It is a basic equation of an artificial neuron (1.1) with regard to a 2D space, but convolutional layers in image processing are built in a particular way that enables the feature extraction. This operation moves through the input and detects the features in the entire image by analogy to (2.2). The visualisation of this process for a two-dimensional filter is shown in the Figure 2.2.

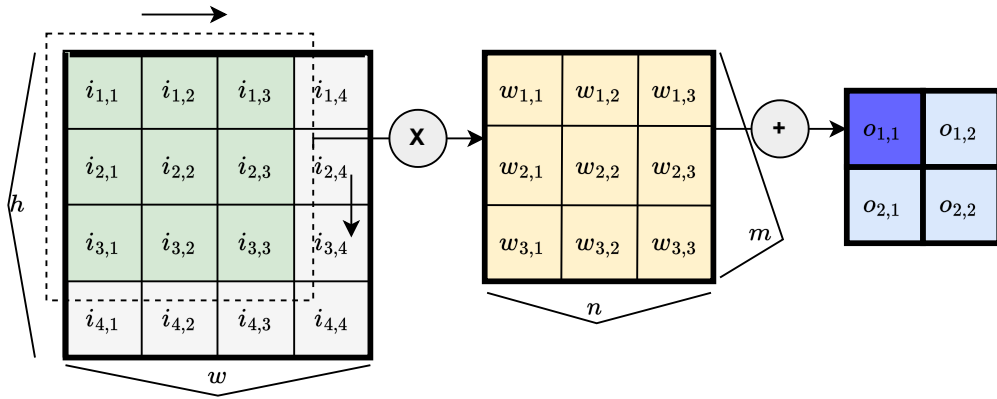


Figure 2.2: Simplified diagram of a convolution operator on an image (bias and activation function are omitted for simplifying the diagram).

Taking the image and filter indices into account based on (2.2) a single value of an output is given by the following equation:

$$o_{p,q} = \varphi \left( \sum_{j=1}^m \left( \sum_{k=1}^n w_{j,k} \cdot i_{y,x} \right) + b \right), \quad (2.6)$$

where  $j$  is a row number, the  $k$  is a column number,  $y = (j + p - \frac{m-1}{2})$ , and  $x = (k + q - \frac{n-1}{2})$ ,  $o_{p,q}$  is an element from output matrix  $\mathbf{O}$ .

A convolution may be computed across multiple channels and multiple filters may be applied on one input, then a single filter is a tensor -  $\mathbf{F} \in \mathbb{R}^{n \times m \times c}$ , where  $c$  is a number of channels, input is a tensor -  $\mathbf{P} \in \mathbb{R}^{w \times h \times c}$ , and all the filters are 4D tensor  $\mathbf{C} \in \mathbb{R}^{n \times m \times c \times f}$ , with  $f$  filters. The output of the convolution layer is a sum of products from each channels as schematically shown in Figure 2.3.

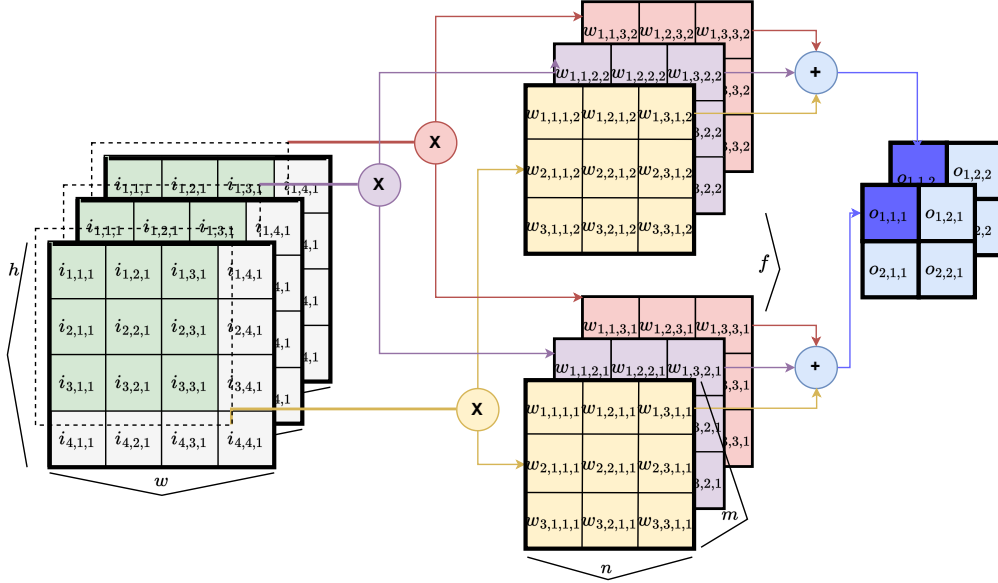


Figure 2.3: Simplified diagram of a convolution operator with multiple filters on a multi-channels input.

The output tensor -  $\mathbf{O} \in \mathbb{R}^{\hat{h} \times \hat{w} \times f}$  in images processing is called a *feature map*, and  $\hat{w}, \hat{h}$  denote its width and height, the final equation with regard to all the parameters is as follows:

$$o_{p,q,f} = \varphi \left( \sum_{j=1}^c \left( \sum_{k=1}^m \left( \sum_{l=1}^n w_{k,l,j,f} \cdot i_{x,y,j} \right) + b_f \right) \right), \quad (2.7)$$

where  $z$  refers to the input channel number,  $f$  to the filter number.

Despite CNN differing from MLP in terms of the connections between layers, there are also many similarities, such as a feed-forward structure and neurons equations, thus the process of adjusting the weights is performed with backpropagation [23].

Based on the previous equation, one can define a number of parameters in one convolutional layer:

$$N_{\text{params}} = (n \cdot m \cdot c + 1) \cdot f, \quad (2.8)$$

and the number of floating point operations to compute an output from a single conventional layer:

$$FLOPs = 2 \cdot \underbrace{(n \cdot m \cdot c + 1) \cdot f}_{N_{\text{params}}} \cdot \underbrace{\frac{(w - \frac{n-1}{2} + P_x)}{\hat{w}}}_{\hat{w}} \cdot \underbrace{\frac{(h - \frac{m-1}{2} + P_y)}{\hat{h}}}_{\hat{h}}, \quad (2.9)$$

output size

when  $S_x, S_y$  denote strides - the number of skipped pixels when the kernel moves through the input data in both directions, and  $P_x, P_y$  are borders padding. The 2 factor is because each parameter has to be multiplied by the input and then summed up. Usually the parameters are chosen as:  $m = n$ ,  $P \triangleq P_x = P_y = \frac{n-1}{2}$ ,  $S \triangleq S_x = S_y$ , hence (2.9) can be simplified to the following form:

$$FLOPs = 2 \cdot (m^2 \cdot c + 1) \cdot f \cdot \frac{w \cdot h}{S^2}, \quad (2.10)$$

**Example** (AlexNet complexity). For a representative example: the first convolutional layer in AlexNet has 96 receptive fields of size  $11 \times 11$ , the size of input is  $227 \times 227 \times 3$ , and the *stride* is 4, thus the number of parameters in this layer is 34944, but the number of floating point operations in this layer is  $0.224 \cdot 10^9$ . For the second layer  $m \times n = 5 \times 5$ ,  $f = 256$ ,  $S = 2$ , so  $N_{params} = 614656$  and  $FLOPs \simeq 0.93 \cdot 10^9$ . This exemplifies the computational complexity of conventional layers. In fact, all the multiplications can be performed in parallel because they are independent, and it is relatively easy to accelerate them on a GPU, however, the computing efficiency needed for the entire convolution is large and can be problematic in some systems.

Knowing the basic concepts of convolutions, and taking into account that one filter functioning as one feature detector, we can consider a fundamental concept of deep learning. Each convolution layer learns to represent more abstract features than the previous one, so the more convolutional layers, the more the level of abstraction. As the level of abstraction increases the feature map matrix typically becomes wider (more channels) although smaller. Adding more channels is a result of using more filters in deeper layers, decreasing the size of the feature map is performed with the use of pooling [66], however using stride for convolutional layers may reduce the size too. Based on this, the first layers in CNN represent basic image features, like edges, corners, etc., then the level of abstraction increases and the network later may represent more compound shapes.

Based on that, the hypothesis in this study is that feeding the first layers with already preprocessed data, that represents more complex information than the raw pixel data may result in reducing a number of convolutional layers in the network without losing its accuracy or actually increasing accuracy when there is a possibility to learn the preprocessing layer on unlabelled data.

## 2.2 UNSUPERVISED FEATURE EXTRACTORS AND AUTOENCODERS

---

Many commonly known methods tackle unsupervised data processing. Some such as clustering [67], data transformations [68, 69], regression [70] or any sort of structured prediction [71] are based on relatively simple techniques which can be efficient for some tasks. However, these methods have not been applied successfully in large scale data classification. One of the major disadvantages is that they do not have a standardised measure that would describe how well a model fits the data, and also their level of abstraction in data representation is rather low. On the other hand, there exist more complex approaches based on neural networks that have similar capabilities but can be larger and trainable in an unsupervised manner. One of the most commonly used models for this task is an autoencoder [72, 73]. A basic form of an autoencoder is presented in Figure 2.4. The architecture is similar to MLP, it is built from an input -  $\mathbf{x} \in \mathbb{R}^n$ , output -  $\mathbf{y} \in \mathbb{R}^n$ , and at least one hidden layer -  $\mathbf{h} \in \mathbb{R}^m$  called sometimes *latent space*, the sizes of input and output signals are the same, although larger than the size of latent space ( $n > m$ ). As an autoencoder is de-facto a standard feed-forward neural network, a backpropagation algorithm may be applied in order to train the parameters as given in (1.8) and (1.9). The loss function can be defined by  $e(\mathbf{y}, \mathbf{x})$ , so the output follows the input during the learning process. Thus  $\mathbf{h}$  vector is simultaneously an image of input  $\mathbf{x}$  and a preimage of output  $\mathbf{y}$ . In other words, an autoencoder is composed of two parts: encoder ( $\mathbf{x} \mapsto \mathbf{h}$ ) and decoder ( $\mathbf{h} \mapsto \mathbf{y}$ ). The number of hidden layers between the input and the latent space is usually the same as between latent space and output, although it is not a strict constraint and unsymmetrical autoencoders exist too [74, 75]. There are many ways to build an autoencoder architecture, but for image processing purposes they may be composed of convolutional layers [76].

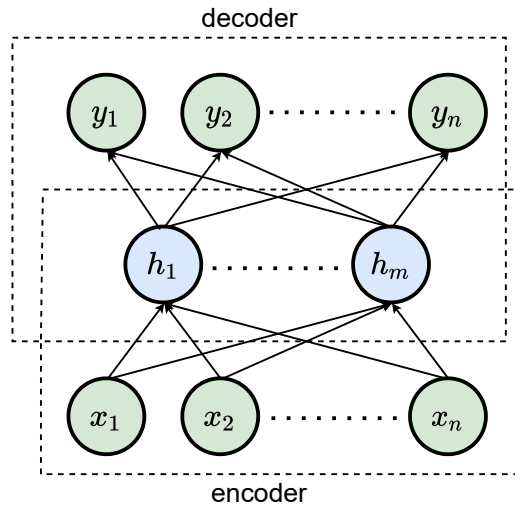


Figure 2.4: Simplified diagram of an Autoencoder.

The main ability of an autoencoder is that it can learn how to represent the high-dimensional data by a space with lower dimensionality and how to reconstruct the input data if it is broken. Nevertheless, an autoencoder is a deterministic model and its generalisation ability is highly limited to data that have been used for training. The major improvement to an autoencoder that tackles this problem is a Variational Autoencoder (VAE) [77], formally a generative model that generalises the autoencoder idea by learning how to generate new data. The way the VAE improves the generalisation ability of an autoencoder is the use of probabilistic distribution to describe the data. In general, VAEs are one of the most important achievements in artificial intelligence, notably in image processing [78]. There exist many variations of VAEs [79], For classification purposes it can be used to pretrain the first convolutional layers, however, VAEs are composed of a large number of parameters, from 8 to 40 million [80], which makes them quite complex, thus it is again a problem in systems with limited resources.

## 2.3 ENERGY-BASED MODELS

Another good example of a neural network that is able to learn in an unsupervised manner is a Hopfield model [81]; it is a recurrent neural network that serves as an auto-associative memory, which means it can learn certain patterns from an input data. In contrast to previously discussed architectures it is a dynamic model, thus there is a time  $t$  associated to network states updates. The network associates some states of its units based on an input

at time  $t = 0$ , then at time  $t = t + T_s$  (where  $t$  denotes the sample number and  $t_s$  denotes sampling time), then changes them dynamically depending on what was computed previously.

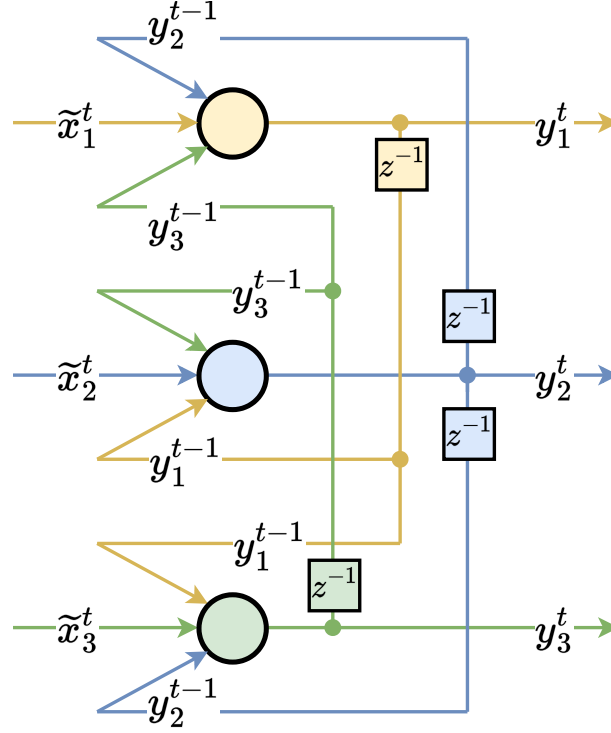


Figure 2.5: Simplified diagram of a Hopfield network.

A simplified form of a Hopfield network is shown in Figure 2.5. It is composed of an input vector  $\tilde{\mathbf{x}}^t = [\tilde{x}_1^t \ \tilde{x}_2^t \ \dots \ \tilde{x}_n^t]^\top$  and an output vector  $\mathbf{y}^t = [y_1^t \ y_2^t \ \dots \ y_n^t]^\top$ , and  $t$  denotes a discrete time. The recurrence is done by using delayed the output  $\mathbf{y}^{t-1}$  as an additional input, so the vector being processed by the network is  $\mathbf{x}^t = (\tilde{\mathbf{x}}^t, \mathbf{y}^{t-1}) = [x_1^t \ x_2^t \ \dots \ x_n^t]^\top$ . In the basic form of Hopfield network the units are bipolar, so they may have two of possible values  $\{-1, +1\}$ . The state of the output  $\mathbf{y}^t$  is computed with the following formula:

$$y_i = \tilde{\varphi} \left( \underbrace{\sum_{j=1}^n w_{i,j} x_j - b_i}_{\triangleq \psi_i} \right), \quad (2.11)$$

where  $b_i$  is a bias of particular neuron, and  $\tilde{\varphi}$  is an activation function:

$$\tilde{\varphi}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}, \quad (2.12)$$

and the weights matrix  $\mathbf{W}$  is:

$$\mathbf{W} = \begin{bmatrix} 0 & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & 0 & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & 0 & \cdots & w_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & w_{n,3} & \cdots & 0 \end{bmatrix}. \quad (2.13)$$

Such a matrix is symmetric ( $\forall_{i,j} w_{i,j} = w_{j,i}$ ), and since  $\forall_i w_{i,i} = 0$  there are no connections between output and input of the same neurons. These weights are trained with the use of Hebbian rule [82].

The Hopfield network is an energy-based model [83], which means that it has a special scalar value associated with its joints configuration called *energy*. This quantity changes during updates of the units and the network's dynamic tends to decrease it. This is in general a feature of a dynamic system in which there can be considered a stability problem. The energy value in a Hopfield network is a deterministic equation and is given as follows:

$$E = -\frac{1}{2} \sum_{i,j} w_{i,j} x_j x_i - \sum_i x_i b_i. \quad (2.14)$$

Based on this, we can prove that the energy will never increase during the neuron state changes. Consider the following equation for a single unit  $i$  energy:

$$E_i \triangleq -\frac{1}{2} \sum_j w_{i,j} x_j x_i - x_i b_i. \quad (2.15)$$

The energy change at time  $t + 1$  can be defined as follows:

$$\begin{aligned} \Delta E_i^{t+1} &= E_i^{t+1} - E_i^t \\ &= \left( -\frac{1}{2} \sum_j w_{i,j} x_j x_i^{t+1} - x_i^{t+1} b_i \right) - \left( -\frac{1}{2} \sum_j w_{i,j} x_j x_i^t - x_i^t b_i \right) \\ &= - \sum_j (w_{i,j} x_j (x_i^{t+1} - x_i^t)) - b_i (x_i^{t+1} - x_i^t) \\ &= - \underbrace{(x_i^{t+1} - x_i^t)}_{\Delta x_i} \underbrace{\left( \sum_j w_{i,j} x_j - b_i \right)}_{\psi_i}. \end{aligned} \quad (2.16)$$

Based on that and according to (2.11) and (2.12) one can write:

$$\begin{aligned}
(x_i^t = -1 \wedge x_i^{t+1} = 1) &\implies \psi_i \geq 0 \\
(x_i^t = 1 \wedge x_i^{t+1} = -1) &\implies \psi_i < 0 \\
x_i^{t+1} > x_i^t &\implies (\Delta x_i > 0 \wedge \psi_i \geq 0) \text{ (neuron enables)} \\
x_i^{t+1} < x_i^t &\implies (\Delta x_i < 0 \wedge \psi_i < 0) \text{ (neuron disables)} \\
x_i^{t+1} = x_i^t &\implies \Delta x_i = 0 \\
\therefore \Delta E_i = -(+)(+) \vee \Delta E_i = -(-)(-) \vee \Delta E_i = 0 &\implies E_i^{t+1} \leq E_i^t. \quad (2.17)
\end{aligned}$$

As a result, (2.16) and (2.17) present a formal proof that the energy cannot increase during the state updates in a Hopfield network. This is important to understand because of the further considerations in this dissertation. No possibility to increase the energy implies no possibility to leave the local energetic minimum, which is the major problem of a standard Hopfield network and results in its limited usability. Schematically it is illustrated in Figure 2.6. Nevertheless, these models were successfully applied to solve many machine learning problems, such as image processing [84, 85], or some numerical computing difficulties [86, 87, 88, 89], also some recent works show its potentially enhanced usability by introducing new learning rules and a new energy function [90].

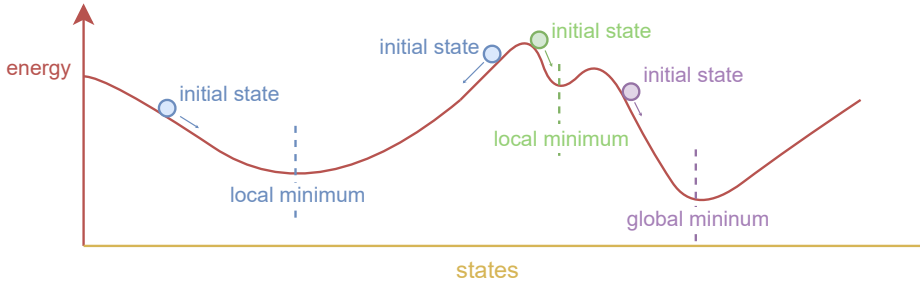


Figure 2.6: Theoretical graph of energy function versus states in a Hopfield network.

A modification to a deterministic Hopfield network is a Boltzmann Machine (BM) [91]. It shares similar concepts, however, by introducing hidden nodes and using stochastic dynamics in the state updates the model does not suffer from tending to local minima. However, BMs in general have serious practical problems with training when their dimensionality increases [92]. As a result, their usability is also limited, but by introducing the restriction that removes the intra-layer connections these concerns may be resolved. The model without connections between neurons in visible layers and without



connections in hidden layers is called Restricted Boltzmann Machine (RBM) [93]. The RBM is a crucial model in this study, so its architecture and its properties will be discussed in detail in the next chapter.

## 2.4 FEATURE DESCRIPTORS

### 2.4.1 Binary descriptors

Binary descriptors were widely utilised before the convolutional feature extractors were applied to image processing problems. The theory behind them is relatively simple, and can be generalised for any kind of descriptor type. Let  $\mathbf{P} \in \mathbb{A}^{h \times w \times c}$  be an input image such as  $\mathbb{A} = \{0, 1, 2, \dots, N\}$ . This image is composed of pixels  $p_{(i,j,:)}$ , where  $i, j$  denote coordinates in an image ( $i \in \{\mathbb{N}_+ : i \leq h\}$ ),  $j \in \{\mathbb{N}_+ : j \leq w\}$ ). For a common implementation of an image  $N = 255$  because of 8-bits image representation, and  $c = 1$  for a greyscale image or  $c = 3$  for an RGB image. A binary descriptor -  $\mathbf{b}$  is a vector of  $k$  bits:  $\mathbf{b} \in \mathbb{B}^k$ , where  $\mathbb{B} = \{0, 1\}$ . A descriptor transformation is a non-linear function  $\zeta$ , that processes a matrix of pixels into a binary string:

$$\zeta : \mathbb{A}^{d \times d \times c} \rightarrow \mathbb{B}^k. \quad (2.18)$$

Formally, a descriptor defines only the  $\zeta$  function and the size  $d$  of the its input matrix being a subtensor of  $\mathbf{P}$ . Instead of using  $d$ , it is more convenient to use  $l = \frac{d-1}{2}$ , which defines the number of pixels neighbouring the centre pixel  $p_{(i,j)}$ . The image subtensor is denoted then by  $\mathbf{P}_{(i,j) \sim l} \in \mathbb{A}^{d \times d \times c}$  and defined as follows:

$$\mathbf{P}_{(i,j) \sim l} \triangleq \begin{bmatrix} p_{i-l,j-l,:} & \cdots & p_{i-l,j+l,:} \\ \vdots & \ddots & \vdots \\ p_{i,j-l,:} & \cdots & p_{i,j,:} & \cdots & p_{i,j+l,:} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{i+l,j-l,:} & \cdots & p_{i+l,j+l,:} \end{bmatrix}, \quad (2.19)$$

and  $\forall_{i,j} p_{(i,j)} \in \mathbb{A}^c$ , so finally one can write that the descriptor provides the following transformation:

$$\zeta : \mathbf{P}_{(i,j) \sim l} \mapsto \mathbf{b}. \quad (2.20)$$

The size of the descriptor is constant for one type of descriptor (sometimes parametrisable), although it varies for different types of descriptors. The intuitive approach to understanding vector  $\mathbf{b}$  is that every single bit in it denotes some part of the particular feature in an image, and the relation

between these bits denotes a particular feature taken from the given part or the images. There are two approaches to gathering the descriptors from the image to make the feature extraction. The first is to take it densely, using every single pixel in the input image as a centre pixel of the transformation or using a grid to skip some pixels. The second approach is to use *keypoints*. A keypoint is a region in the image, that represents some special features, regarded by the processing algorithm as an interesting region. The keypoints selection may be done automatically [94], or even the binary descriptors can be used to decide if a region of a given image is interesting or not (by analysis of the relations between the bits).

If the image transformation is done densely the output size is constant  $\lfloor \frac{h}{s} \rfloor \times \lfloor \frac{w}{s} \rfloor$ , assuming  $s$  is a stride size of the same meaning as in the case of convolutions. For descriptors taken with keypoints the output size varies depending on a number of detected keypoints, that in fact reduces the output dimensionality to have only important descriptors but it may result in problems with further processing since most of the commonly used classifiers require constant size of an input.

There are many types of binary descriptors that perform the  $\zeta(\cdot)$  function in different ways. Among others the most popular ones are Fast Retina Keypoint (FREAK) [95], Local Binary Pattern (LBP) [96], Binary Robust Invariant Scalable Keypoint (BRISK) [97], Features From Accelerated Segment Test (FAST) [98], Binary Robust Independent Elementary Features (BRIEF) [99], Oriented Fast and Rotated Brief (ORB) [100]. The binary descriptors played important role in machine vision tasks, only to mention some of them it was image classification [101, 102], object detection [103, 104, 105] or panorama stitching [106].

### 2.4.2 Local Binary Pattern

---

Local Binary Pattern is one of the simplest binary descriptors. In [107] it has been shown that LBP can be an efficient tool in a preprocessing layer. Such a descriptor constitutes a key element of the method proposed in this thesis. In this section details of this type of descriptor will be discussed. The theoretical background of the LBP is simple, it performs a number of comparisons between the centre pixel and its neighbouring pixels, where the input image is in greyscale. The size of a neighbourhood -  $l$  is the parameter that defines the sampling pattern size and as a result the size of the output binary string -  $k$ , the notion for naming the descriptor including its size in this study is  $LBPk$ . The sampling pattern is a circle, hence the neighbourhood can be defined by a radius -  $r$  (in pixels), and the pixels being used for computing the output vector are obtained by checking if the circle intersects

a given pixel. An example of two configurations of LBP is presented in Figure 2.7.

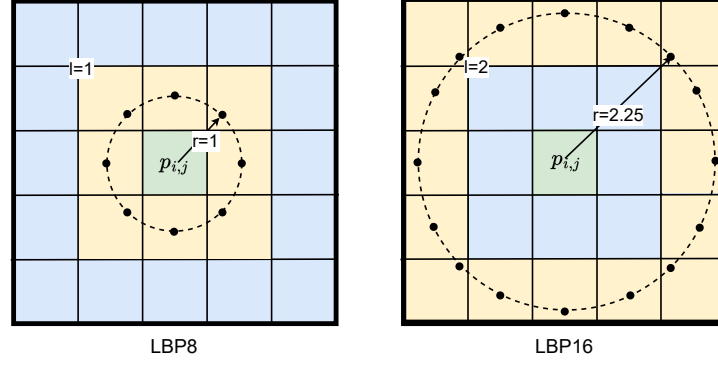


Figure 2.7: Sampling patterns for two example LBP configurations. Green is a centre pixel, yellow denotes neighbouring pixels taken for computing the descriptor, blue pixels are omitted by the descriptor.

From this Figure one can easily conclude that the  $l$  for  $\mathbf{P}_{(i,j)\sim l}$  can be defined as:

$$l = \lfloor r \rfloor. \quad (2.21)$$

Then, a single bit of a binary vector  $\mathbf{b} = \zeta_{LBP}(\mathbf{P}_{(i,j)\sim l}) = [b_1 \ b_2 \ \dots \ b_k]^\top$ , is:

$$b_x = \alpha(p_{i,j} - p_{\lambda(x,i,j)}), \quad (2.22)$$

where  $\alpha$  is defined as follows:

$$\alpha(s) = \begin{cases} 1 & \text{for } s > 0 \\ 0 & \text{for } s \leq 0 \end{cases}, \quad (2.23)$$

and  $\lambda(x, i, j)$  defines the coordinates of neighbouring pixel for a comparison with centre pixels ( $\lambda : \mathbb{N}^3 \rightarrow \mathbb{N}^2$ ). Hence this may be an ambiguous parameter of an LBP, the implementation used in this study takes the top-left corner of the neighbourhood ( $y = i - l, z = j - l$ ) for  $x = 0$ , then the next neighbouring

pixels are chosen clockwise, therefore for example for LBP8:

$$\lambda(x, i, j) = \begin{cases} [i - 1 & j - 1] & \text{for } x = 0 \\ [i - 1 & j + 0] & \text{for } x = 1 \\ [i - 1 & j + 1] & \text{for } x = 2 \\ [i + 0 & j + 1] & \text{for } x = 3 \\ [i + 1 & j + 1] & \text{for } x = 4 \\ [i + 1 & j + 0] & \text{for } x = 5 \\ [i + 1 & j - 1] & \text{for } x = 6 \\ [i + 0 & j - 1] & \text{for } x = 7 \end{cases} \quad (2.24)$$

or in a more compact form:

$$\lambda(x, i, j) = \left[ i + \left[ \cos \left( (x - 5) \frac{2\pi}{8} \right) \right] j + \left[ \sin \left( (x - 5) \frac{2\pi}{8} \right) \right] \right]. \quad (2.25)$$

Some of the use cases may convert the  $\mathbf{b}$  vector to an integer value -  $v$  with the following formula:

$$v = \sum_{i=1}^k (b_i \cdot 2^{i-1}), \quad (2.26)$$

but for the case of processing the descriptor by a neural network, this is not very useful, despite reducing the output dimensionality, it does not treat changes of every bit equally. In other words, converting  $\mathbf{b}$  to  $v$  causes some bits to affect the output more than others.

To demonstrate the sensitivity on particular features one can consider the simplest LBP8 descriptor for which the image subregion is defined by the following matrix:

$$\mathbf{P}_{(i,j)\sim 1} = \begin{bmatrix} p_{i-1,j-1} & p_{i-1,j} & p_{i-1,j+1} \\ p_{i,j-1} & p_{i,j} & p_{i,j+1} \\ p_{i+1,j-1} & p_{i+1,j} & p_{i+1,j+1} \end{bmatrix}, \quad (2.27)$$

Then the  $\mathbf{b}$  vector is defined as follows:

$$\zeta_{LBP8}(\mathbf{P}_{(i,j)\sim 1}) = [\alpha(p_{i,j} - p_{i-1,j-1}) \alpha(p_{i,j} - p_{i-1,j+0}) \\ \alpha(p_{i,j} - p_{i-1,j+1}) \alpha(p_{i,j} - p_{i+0,j+1}) \\ \alpha(p_{i,j} - p_{i+1,j+1}) \alpha(p_{i,j} - p_{i+1,j+0}) \\ \alpha(p_{i,j} - p_{i+1,j-1}) \alpha(p_{i,j} - p_{i+0,j-1})]^\top \quad (2.28)$$

Hence, after LBP8 transformation,  $\mathbf{b}$  is an 8-bits vector where each bit denotes a comparison result of the centre pixel with different neighbouring

pixel. The visualisation of some trivial  $3 \times 3$  patterns and their LBP8 outputs is shown in Table 2.1 where we can observe how the particular bit of the output vector changes depending on the feature that is being processed. LBP8 can code  $2^8$  types of patterns, which is a relatively low number, but for practical purposes, this may be enough because those vectors represent much more abstract features than raw pixel representation, which is also usually an 8-bits number.

	pattern 1 corner	pattern 2 horizontal line	pattern 3 vertical line	pattern 4 point	pattern 5 flat region
visualisation					
<b>b</b>	[11101011]	[11101110]	[10111011]	[11111111]	[00000000]

Table 2.1: Example  $3 \times 3$  patterns and their LBP8 output.

Local Binary Pattern is a basic image region descriptor, but this simplicity can be seen as a significant advantage. It can be implemented as a very fast image transformation. It is worth noticing that all the pixels can be processed in parallel, so the use of GPU or CPU multithreading will result in decreasing the time of processing. LBP descriptor has been successfully utilised in some image processing problems, among others it achieved a good performance in facial recognition [108, 109].

### 2.4.3 Real-valued feature descriptors

As an alternative to binary descriptors, we can discuss feature extractors that use real-valued feature vectors, like Scale Invariant Features (SIFT) [110], Speed Up Robust Features (SURF) [111], Histogram of Oriented Gradient [112]. They can be formalised by:  $\mathbf{b} \in \mathbb{R}^k$ , and then the remainder of the equations remain the same as it was for binary descriptors. The real-valued descriptors are mentioned because of their general importance in machine

vision [113, 114, 115, 116, 117], but they will not be analysed in this dissertation, since they do not meet the condition of our preprocessing layer, where the intermediate stage of processing is binary. The second reason is that they are slower and more memory-consuming. To store one feature, real-valued descriptor needs 32 times more memory than in the case of binary vectors (assuming they are kept as float type). To illustrate this issue in more detail, a simple experiment was performed, which measured the averaged execution time of  $\zeta(\cdot)$  for some common binary and real-valued descriptors. The results are shown in Table 2.2, we notice that all the real-valued descriptors are slower, this is because the  $\zeta(\cdot)$  function for binary transformers relies mostly on comparisons while others rely on gradients. Thus, as the dissertation focuses on the speed of processing, the binary vectors are the better choice. For clarification, the time of response may depend on the parameters of a given descriptor, but for the experiments, they were set as default, hence the general dependency showing binary ones as faster should be clear enough.

descriptor type	real-valued			binary			
	SIFT	SURF	HOG	BRIEF	FREAK	BRISK	LBP8
time	1	0.48	0.53	0.05	0.32	0.33	0.0003

Table 2.2: Average response times of different descriptors for a single key-point, times are relative to the slowest descriptor (SIFT).

#### 2.4.4 Feature descriptors aggregation

As mentioned before, for classification purposes the output size of the feature descriptor layer varies depending on how many keypoints have been found, the output can be regarded as a matrix of size  $N_v \times k$ , where variable  $N_v$  is the number of located keypoints, so to pass the descriptors to a classifier like SVM or MLP, some form of aggregation is needed. To define the aggregation one can assume that this is a process of reducing output matrix dimensionality to be constant independently on the input image and number of keypoints. This is important in terms of the considerations in this study since the discussed classification architectures rely on constant input size and aggregation is required.

For real-valued descriptors, there are many commonly known aggregation techniques. Among others, the most popular one is Bag of Words (BoW) [118] encoding, however, the BoW method has some limitations, because it requires the data sample to fit the codebook, which is done with *k-means* clustering, typically with large  $k$  ( $\approx 10\text{K}-1\text{M}$ ). Furthermore, it requires ap-

proximate nearest neighbour methods to fit the codebook [119]. Then having the constant number -  $k_c$  of the feature vector it is possible to form a histogram of those features occurrences. As a result, the image is represented by a histogram. The nature of a feature descriptor when each value may denote something completely different does not allow the use of histogram binning, thus the number of bins in the histogram has the same size as the codebook -  $k_c$  and is very sparse [107]. Some extensions to this algorithm have been properties, such as, VLAD [120] or Fisher Vector Encoding [121]. They use first (VLAD) or second-order (Fisher) differences with the code-words. Unfortunately, the usability of BoW methods is limited to real-valued feature vectors. The limitation is mainly related to the algorithm of generating the codebook that relies on k-means that uses Euclidean distances to compare the vectors, which is not applicable to binary vectors. Comparing binary vectors is usually performed with Hamming distance [122], but averaging multiple binary vectors produces a non-binary vector for which the Hamming distance cannot be used [107].

Another method is to create a histogram directly from the extracted features by converting the binary strings to  $v$  values as shown in (2.26). Then the binary vectors become a single integer value, and the histogram representation can be formed by counting the occurrences of all possible integers. It is easy to count the maximum  $v$  value and so the number of bins in the histogram is  $2^k$ , so there is a clear limitation of this method for feature descriptors that utilise large binary strings. For example, FREAK and BRISK use  $k = 512$  which makes this method impossible to apply - we still cannot use binning and the theoretical histogram for these cases would be  $2^{512}$ . On the other hand, the LBP descriptor can have a very small feature vector. For example, the use of LBP8 will result in having  $2^8 = 256$  bins, which is a very small number, easily processed and stored in memory. In [108] it was shown how to successfully apply this to facial recognition problems, however, the authors expanded the histogram so that it became a concatenation of histograms from different regions.

Although some techniques can be applied for the aggregation of feature vectors, all of them have a number of limitations, principally in terms of binary descriptors. This has been also approached in this study, a neural network that processes the binary pattern can be used to either aggregate the binary strings gathered from keypoints or for densely gathered descriptors to create a histogram where the binning is applicable, so its size is configurable. However, the histogram is used only for image comparison, not for image classification. For classification purposes, one may use another ability of the neural network which aggregates the descriptors and allows utilising it by a classifier by computing their averaged distance to a training dataset, which

is a novel method proposed by the author and will be discussed in detail in the following chapters.

## 2.5 ENHANCED LOCAL BINARY PATTERN - COLOUR LBP8

---

Converting an image from its 3 channels representation (mostly RGB or YUV) to one greyscale channel results in the loss of the information regarding the colour. The conversion may be done in several ways, but the most common is computing the average of the RGB channels  $P_{\text{grey}} = \lfloor \frac{R+G+B}{3} \rfloor$ , or taking the weighted average  $P_{\text{grey}} = \lfloor 0.299R + 0.587G + 0.114B \rfloor$  that is formally the  $Y$  channel from YUV format.

Objects being classified by recognition systems are mainly defined by their shapes, not by their colours, however, colours in vision also play an important role. As a practical example, we do not need to know the colour of object if we desire to distinguish between an animal and a vehicle, but to distinguish the animal species the colour may be very useful, especially as in image processing the size context may be unknown. For example, ImageNet contains more than 10 types of snakes that have similar shapes but differ in colours. The authors of [123] showed that the accuracy of CNN drops by 3% on ImageNet when the input images are converted to greyscale. That is why there is a great need to take the colours into account when the features are being extracted. In this research, it has been addressed by enhancing a regular greyscale descriptor by colour information.

Another concern, for example in the LBP descriptor, is that the intensity of greyscale pixels is also lost, so the processing does not have the information regarding the brightness of a region, only the shape is coded. Intuitively, we can define that to recognise an object correctly we need to have information about its shape and colour, but as the shape is more important, the colour may be defined approximately.

It has been demonstrated previously that the LBP8 codes the shapes in eight bits. The author proposed a novel method to enhance this descriptor by adding additional eight bits to take the colour and brightness into account. The idea is to use comparison between all of three possible pairs of colours ( $RG$ ,  $BR$ ,  $GB$ ) of the centre pixel  $p_{(i,j)}$ . For a single pair  $C_0 - C_1$  we can define 2 bits vector  $\mathbf{d}_{C_0, C_1} = [b_0 \ b_1]$  that describes the colour dependency and is computed as follows.

$$\begin{aligned} b_0 &\triangleq (C_1 > C_2), \\ b_1 &\triangleq (C_2 > (C_1 + T)) \vee ((C_1 < (C_2 + T)) \wedge (C_1 > C_2)). \end{aligned} \quad (2.29)$$



To illustrate the idea, we can show a distribution of a single colour pair with an assumption that the values of both colours are distributed uniformly as in Figure 2.8. This assumption can be taken for an unknown image dataset and the neural network should learn to a particular distribution based on the output from the descriptor.

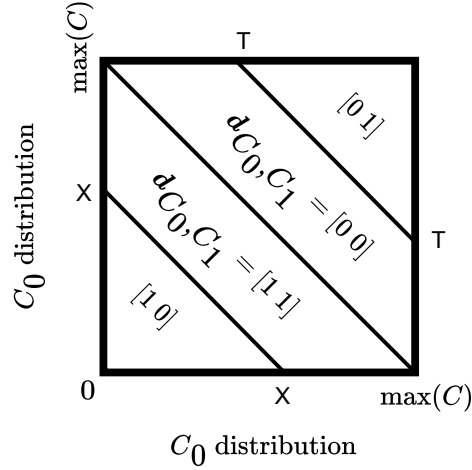


Figure 2.8: Two bits descriptor distribution for a single colours pair.

The value of  $T$  is a threshold and should be computed to equalise the occurrences of each possible  $d$  value over all the possible colour pairs, thus the areas of each figure given by the particular descriptor value is the same:

$$\begin{aligned} \frac{1}{2}X^2 &= \frac{1}{4} \max(C) \therefore X = \frac{\sqrt{2}}{2} \max(C) \\ T &= \max(C) - X \therefore T = \frac{2 - \sqrt{2}}{2} \max(C) \end{aligned} \quad (2.30)$$

Assuming  $C \in \{\mathbb{N} : C \leq 255\}$ ,  $T = 73$ . By using the 3 comparisons between all the possible pairs we can define a 6 bits vector that describes the dependency between the colours. To describe the brightness of the centre pixel the simplest solution is binary shifting, thus we can add another 2 bits -  $\mathbf{d_I} = [b_0 \ b_1]$  to the descriptor:

$$\begin{aligned} b_0 &= P_{\text{grey}} \gg 7 \\ b_1 &= P_{\text{grey}} \gg 6, \end{aligned} \quad (2.31)$$

and finally the 16 bits enhanced Local Binary Pattern descriptor (called further as CLBP) can be defined by concatenating all of these as shown in Figure 2.9.



Figure 2.9: Colour LBP descriptor.

The first part of the CLBP is a regular LBP8 binary vector, the second is a vector representing the colour. Some example colours with their associated codes are shown in Figure 2.10. The dependency between the colour and its code can be easily recognised. For example first 6 bits in a red colour is coded to [100100], while in a green colour these bits are [010010], because of using the intensity descriptor in the last two bits we can also distinguish between the brightness of these colours (dim green has [01001000] code, while bright green has [01001001] code).



Figure 2.10: Example colours with their 8 bits colour descriptors (white rectangles), and RGB hex codes.

The colours could be coded by binary shifting, but this would cause losing

the dependency between the colours, for example, very dim green #002200 would have same code as code very dim red #220000 or blue #000022. In the proposed solution these colours have still different descriptors because of the comparisons between the channels. Another idea is to perform an LBP8 operation on each of 3 image channels and then concatenate the outputs, but it results in a larger - 24 bits descriptor and represents the shapes in each channel, not the overall feature and its colour. As a result, the CLBP is the basic example of a descriptor that was previously declared as a desired feature extractor, which means its binary vector contains the image region shape and the approximate colour. It will be used in the preprocessing pipeline in a further part of this study, and the suitability of this idea will be shown in experiments.

It is worth mentioning that these additional eight bits describing the colour could be potentially added to another type of binary descriptors. However, as mentioned previously LBP8 is considered as the most robust in terms of what this dissertation focuses on, thus the modifications of other descriptors may be investigated in a further research.



---

## RESTRICTED BOLTZMANN MACHINE

---

The Boltzmann Machine and the Restricted Boltzmann Machine were initially mentioned in the previous chapter as an improvement to a Hopfield Network, and formally both can be classified as a Hopfield network variation. The improvement was achieved by changing the paradigm from deterministic to stochastic and by introducing hidden units which results in a possibility to leave local energetic minima. This chapter focuses on explaining how they work in terms of unsupervised learning and data processing. It also provides a short explanation of how the RBM is implemented for research purposes.

### 3.1 OVERVIEW OF AN RBM

---

The model used in this study is an RBM, but for historical reasons, it is worth beginning the analysis with a short introduction of the BM as it was invented first, and the RBM is technically only an improvement on the BM that resolves some training problems.

The BM was popularised in the 1980's by Hinton and Sejnowski and in general, it differs from a Hopfield Network mainly in two factors:

- the BM has hidden units while Hopfield network composes only from visible units,
- the BM is a probabilistic model while Hopfield network is a deterministic model.

Schematically, the BM can be presented as in Figure 3.1. The weights between the nodes in the BM do not generate the output value directly, but describe the probability with which the output reaches a value of 1. That makes the entire dynamics of this model to be stochastic. The visible units denote those neurons that can be directly observed and where the input is

loaded, the hidden units have a similar meaning to the latent space in an autoencoder as they are a representation of input in another space. The energy function remains the same as it was for a Hopfield Network, however it expands to the form of two types of units:

$$\begin{aligned}
 E &= -\frac{1}{2} \sum_{i,j} w_{ij} x_j x_i - \sum_i x_i b_i \\
 &= - \left( \sum_{i,j \in VV} w_{ij} v_j v_i + \sum_{i,j \in HH} w_{ij} h_j h_i + \sum_{i,j \in VH} w_{ij} v_j h_i \right) \\
 &\quad - \left( \sum_i v_i b_{vi} + \sum_i h_i b_{hi} \right),
 \end{aligned} \tag{3.1}$$

where  $VV$ ,  $HH$ ,  $VH$  denote respectively the connections between visible-visible, hidden-hidden, visible-hidden units. However, because of the stochastic dynamics, BMs can leave the local energy minima which was the major concern of Hopfield Networks.

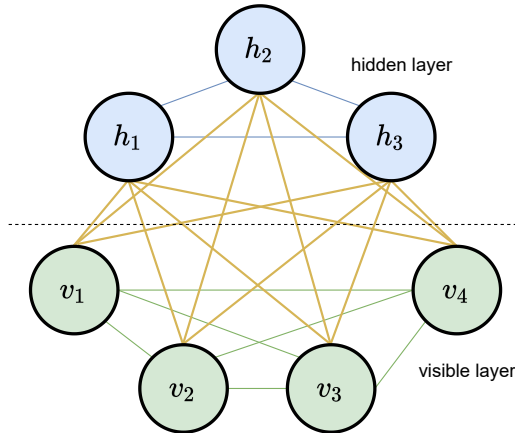


Figure 3.1: Simplified diagram of a Boltzmann Machine. Stochastic dynamics is skipped for simplicity.

As mentioned earlier, BMs are not very useful for practical problems, especially where their dimensionality is large. Thus, there was a need to train them efficiently. RBMs solve this problem. The RBM was invented by Smolensky in 1986 as a simplification of the BM, however the efficient training algorithm was not available until 2002, when Hinton invented a very fast training method, this algorithm will be discussed in detail after introduction to the entire structure of an RBM.

The importance of RBMs in machine learning is high, they were proven to be effective in many complex problems such as feature extraction [124], dimensionality reduction [125], classification [126], or collaborative filtering [127]. Despite today's CNNs outperforming RBMs in image processing, this study shows that there is still a research gap where RBMs may play an important role. This is because of some interesting properties that RBMs have, and other models do not. These properties are explained in this chapter and their usability in machine vision is presented in the next chapter, however among others, the most essential ones are that an RBM can be trained in a fully unsupervised manner, it can detect inherent patterns in training data [128, 129, 130], and then reconstruct them or assign a probability of occurrence to a given input. In this research, an RBM is used as a neural network that processes the binary input which is a binary feature descriptor, and this chapter focuses only on these RBM properties that are needed for this processing task.

Foremost, the restriction that distinguishes RBMs from BMs should be clarified, it is nothing else than just an elimination of intra-layer connections, so by analogy to Figure 3.1, there are no blue and green connections. It can be schematically shown as in Figure 3.2 and to emphasise the dynamics, an unrolled version of an RBM is shown in Figure 3.3. Based on that, we can define some parameters and signals in this model:

- $R_v$  - number of visible units, defined by the size of input and: ,
- $R_h$  - number of hidden units, a hyperparameter defined by user,
- $\mathbf{W}$  - weights matrix:  $\mathbf{W} \in \mathbb{R}^{R_h \times R_v}$ ,
- $\mathbf{a} = [a_1 \ a_2 \ \dots \ a_{R_v}]^\top$  - biases vector in visible layer:  $\mathbf{a} \in \mathbb{R}^{R_v}$ ,
- $\mathbf{b} = [b_1 \ b_2 \ \dots \ b_{R_h}]^\top$  - biases vector in hidden layer:  $\mathbf{b} \in \mathbb{R}^{R_h}$ ,
- $\mathbf{v} = [v_1 \ v_2 \ \dots \ v_{R_v}]^\top$  - visible units  $\mathbf{v} \in \{0, 1\}^{R_v}$ ,
- $\mathbf{h} = [h_1 \ h_2 \ \dots \ h_{R_h}]^\top$  - hidden units  $\mathbf{h} \in \{0, 1\}^{R_h}$ .

An RBM is defined then, by total number of parameters =  $R_v \cdot R_h + R_v + R_h$ . Considering the only two possible states - 0 or 1 of each unit from  $\mathbf{v}$ ,  $\mathbf{h}$  and the independence between each unit in intra-layer as well as the dependence only on the previous states we can also claim that RBM dynamics is an example of Bernoulli's process.

---

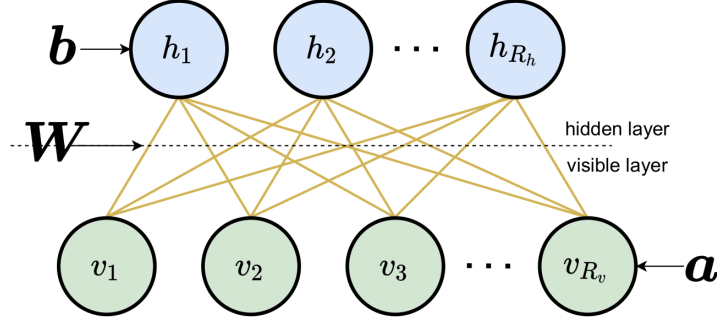


Figure 3.2: Simplified diagram of a Restricted Boltzmann Machine.

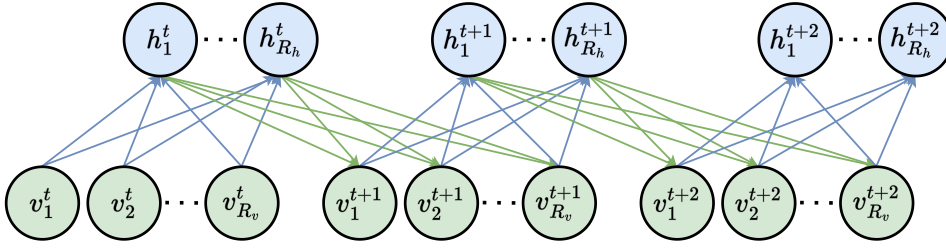


Figure 3.3: Simplified diagram of an unrolled Restricted Boltzmann Machine.

As an RBM is a variation of a Hopfield Network it is a recurrent neural network. The recurrence is performed by computing the hidden and visible states alternately, thus output from a current step becomes an input to the next step. The first step in that process is to update hidden units with given visible units values (input). The conditional probability that the given hidden neuron value is equal to 1 is given as follows:

$$P(h_j = 1 | \mathbf{v}) = \sigma \left( \underbrace{b_j + \sum_{i=1}^{R_v} w_{ij} v_i}_{\triangleq \psi_{h_j}} \right), \quad (3.2)$$

where  $\sigma(\cdot)$  is a sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (3.3)$$

This activation function causes the probability of a neuron being enabled is higher for  $\psi_{h_j} > 0$  and being disabled for  $\psi_{h_j} < 0$  at current time step. Technically, the neuron state activation can be implemented by comparing



the probability value with a number taken from a uniform distribution with the use of the following equation:

$$h_j = P(h_j = 1 \mid \mathbf{v}) > X_{\sim\mathcal{U}}. \quad (3.4)$$

The next step is to update the visible units based on the output from previously computed hidden states, this is done analogously:

$$P(v_i = 1 \mid \mathbf{h}) = \sigma \left( \underbrace{a_i + \sum_{j=1}^{R_h} w_{i,j} h_j}_{\triangleq \psi_{v_i}} \right), \quad (3.5)$$

$$v_i = P(v_i = 1 \mid \mathbf{h}) > X_{\sim\mathcal{U}}. \quad (3.6)$$

Both steps can be simplified to a matrix notation to obtain the probabilities of entire visible and hidden vectors.

$$\begin{aligned} \mathbf{p}_h &= [P(h_1 = 1 \mid \mathbf{v}) \ P(h_2 = 1 \mid \mathbf{v}) \ \dots \ P(h_{R_h} = 1 \mid \mathbf{v})]^\top = \mathbf{b} + \mathbf{W}\mathbf{v}, \\ \mathbf{h} &= \mathbf{p}_h > X_{\sim\mathcal{U}}^{R_h}, \\ \mathbf{p}_v &= [P(v_1 = 1 \mid \mathbf{h}) \ P(v_2 = 1 \mid \mathbf{h}) \ \dots \ P(v_{R_v} = 1 \mid \mathbf{h})]^\top = \mathbf{a} + \mathbf{W}^\top \mathbf{h}, \\ \mathbf{v} &= \mathbf{p}_v > X_{\sim\mathcal{U}}^{R_v}. \end{aligned} \quad (3.7)$$

After the second step, the generated vector  $\mathbf{v}$  called *reconstruction* should be similar to the one that initialised the first iteration. Then the process can be repeated with the reconstruction used as an input vector to compute the next hidden vector. The entire process for  $\mathbf{v}$  given as input is presented as Algorithm 1.

---

**Algorithm 1** RBM data processing
 

---

```

step ← 0;
while step < number of steps do
    ph ← σ(Wv + a);
    h ← ph > X~URh;
    pv ← σ(W⊤h + b);
    v ← pv > X~URv;
    step ← step + 1;
end while
    
```

---

There are several important things to note. The first is that (3.2) and (3.5) are the standard neuron equations with a sigmoid activation function describing the probability of a neuron being active, and with the use of (3.7) these

are easily implementable in modern libraries supporting matrix arithmetic. The second is that the matrix  $\mathbf{W}$  is shared between the steps, this is one of the differences between RBMs (undirected graphs) and autoencoders (directed graphs) where the weights *input – latent*, and *latent – output* differ. The third is that as a result of using the stochastic dynamics each update of the states may result in different outputs, which makes an RBM a generative model, which means that RBM can generate new data from a given input by learning the probability distribution of the training data. The last is that each update of units depends only on the previous steps, and due to this property, an RBM meets the condition of being a Markov Random Field (MRF) [131, 132], also each iteration can be regarded as a Gibbs sampling which is an algorithm for producing samples from joint probability distribution [133].

The energy function for an RBM is defined the same way as it was for a BM in (3.1), but due to the restrictions it is simplified to the following form:

$$E(\mathbf{v}, \mathbf{h}) = - \sum_{i=1}^{R_v} a_i v_i - \sum_{j=1}^{R_h} b_j h_j - \sum_{i=1}^{R_v} \sum_{j=1}^{R_h} h_j w_{ij} v_i. \quad (3.8)$$

Equivalently, in the matrix notation one can write:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^\top \mathbf{v} - \mathbf{b}^\top \mathbf{h} - \mathbf{h}^\top \mathbf{W} \mathbf{v} \quad (3.9)$$

An RBM also has another interesting property, that with the use of energy it is possible to compute the probability of occurrence given  $\mathbf{v}, \mathbf{h}$  pair, and it is defined with the following formula:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})}, \quad (3.10)$$

where:

$$Z = \sum_{i=0}^{2^{(R_v)}-1} \sum_{j=0}^{2^{(R_h)}-1} (e^{-E(\mathbf{v}_i, \mathbf{h}_j)}) \quad (3.11)$$

and:

$$\mathbf{v}_i = [(v_1 = i \gg R_v - 1 \wedge 1) (v_2 = i \gg R_v - 2 \wedge 1) \cdots (v_0 = i \gg 1 \wedge 1)],$$

$$\mathbf{h}_j = [(h_1 = j \gg R_h - 1 \wedge 1) (h_2 = j \gg R_h - 2 \wedge 1) \cdots (h_0 = j \gg 1 \wedge 1)].$$

$Z$  is called a partition function sums  $e^{-E(\mathbf{v}, \mathbf{h})}$  over all possible  $\mathbf{v}, \mathbf{h}$  configurations to scale the sum of all possible  $P(\mathbf{v}, \mathbf{h})$  to 1. That makes it quite complex, and the partition is not trackable for high-dimensional RBMs and

has to be estimated [134, 135]. It is also possible to define the conditional probability of a configuration of the visible units, given a configuration of the hidden units:

$$P(\mathbf{v} | \mathbf{h}) = \prod_{i=1}^{R_v} P(v_i | \mathbf{h}). \quad (3.12)$$

and analogously for the hidden units, given visible units:

$$P(\mathbf{h} | \mathbf{v}) = \prod_{j=1}^{R_h} P(h_j | \mathbf{v}), \quad (3.13)$$

both equations satisfy the requirement of the Markov Random Field of being independent on history.

Sometimes there may be a need to directly compute the marginal probability  $P(\mathbf{v})$ . This is intuitively simple because it is a sum of all probabilities over possible  $h$  values:

$$P(\mathbf{v}) = \sum_{j=0}^{2^{R_h}-1} (P(\mathbf{v}, \mathbf{h}_j)), \quad (3.14)$$

However, it can be simplified with the use of free energy - the energy of a given visible vector that it would need to have in order to have the same probability as all the possible configurations  $(\mathbf{v}_i, \mathbf{h})$  [136]:

$$e^{-F(\mathbf{v})} = \sum_{j=0}^{2^{R_h}-1} (e^{-E(\mathbf{v}, \mathbf{h}_j)}), \quad (3.15)$$

where  $F(\mathbf{v})$  can be defined with the following formula[136]:

$$F(\mathbf{v}) = - \sum_{i=1}^{R_v} v_i a_i - \sum_{j=1}^{R_h} \log(1 + e^{\psi_{h_j}}). \quad (3.16)$$

Finally, it is possible to compute the marginal probability of a given  $\mathbf{v}$  vector with the use of free energy:

$$P(\mathbf{v}) = \frac{1}{Z} e^{-F(\mathbf{v})}. \quad (3.17)$$

This equation is highly important in the context of the considerations in this dissertation because with the use of it we can compute the probability that a given binary vector given as  $\mathbf{v}$  occurred in the dataset that was used

---

for training. Other models (even the recurrent ones) do not have metrics like that one defined in (3.17), which leads to the conclusion that RBMs may be used not only as a regular network to detect some patterns but also to infer the similarity of a given input to those that were seen during the training phase. The possibility of using matrix arithmetic, makes this process very simple and relatively computationally inexpensive.

Although RBM provides a method of computing the conditional probability, as mentioned earlier, the  $Z$  partition function can be problematic since the computational complexity increases exponentially as the dimensionality of an RBM increases. However, on the other hand, its value has to be computed only once for particular RBM parameters. Also, sometimes its computation may be omitted, for example, if there is no need to know the  $P(\mathbf{v})$  directly just need to compare two probabilities of occurrences of two patterns ( $\mathbf{v}_0$  and  $\mathbf{v}_1$ ). This can be computed with the use of their free energy, the formal explanation is presented as follows:

$$\begin{aligned} \kappa &\triangleq \frac{P(\mathbf{v}_0)}{P(\mathbf{v}_1)} = \frac{\frac{1}{Z}e^{-F(\mathbf{v}_0)}}{\frac{1}{Z}e^{-F(\mathbf{v}_1)}} = e^{F(\mathbf{v}_1)-F(\mathbf{v}_0)} \\ \therefore F(\mathbf{v}_1) > F(\mathbf{v}_0) &\implies \kappa > 1 \implies P(\mathbf{v}_0) > P(\mathbf{v}_1), \\ F(\mathbf{v}_1) < F(\mathbf{v}_0) &\implies \kappa < 1 \implies P(\mathbf{v}_0) < P(\mathbf{v}_1) \end{aligned} \tag{3.18}$$

This is a use case when there is a need to compare two binary patterns in order to know which one occurred in a training dataset more frequently and it is clearly observable that the higher free energy of  $\mathbf{v}_k$  denotes the lower probability of  $\mathbf{v}_k$ .

It was previously mentioned that due to the use of stochastic dynamics in RBM, the energy does not stick unconditionally in local energy minima. The energy for a single hidden unit at time  $t$  can be represented by:

$$E_j^t \triangleq - \left( \sum_i a_i v_i + b_j h_j^t + h_j^t \sum_i w_{ij} v_i \right) \quad (3.19)$$

Computing a change of energy in subsequent steps we have

$$\begin{aligned} \Delta E &= E^{t+1} - E^t \\ \Delta E &= \sum_i a_i v_i + b_j h_j^t + h_j^t \sum_i w_{ij} v_i - \sum_i a_i v_i - b_j h_j^{t+1} - h_j^{t+1} \sum_i w_{ij} v_i \\ \Delta E &= b_j h_j^t - b_j h_j^{t+1} + h_j^t \sum_i w_{ij} v_i - h_j^{t+1} \sum_i w_{ij} v_i, \end{aligned} \quad (3.20)$$

Finally,  $\Delta E$  can be represented by:

$$\Delta E = \underbrace{(h_j^t - h_j^{t+1})}_{-\Delta h_j} \underbrace{\left( b_j + \sum_i w_{ij} v_i \right)}_{\psi_{h_j}}. \quad (3.21)$$

$h_j^t = 0, h_j^{t+1} = 1 \implies -\Delta h_j < 0$		$h_j^t = 1, h_j^{t+1} = 0 \implies -\Delta h_j > 0$	
$\psi_{h_j} > 0$	$\psi_{h_j} < 0$	$\psi_{h_j} > 0$	$\psi_{h_j} < 0$
$P(h_j^{t+1} = 1) > \frac{1}{2}$	$P(h_j^{t+1} = 1) < \frac{1}{2}$	$P(h_j^{t+1} = 0) < \frac{1}{2}$	$P(h_j^{t+1} = 0) > \frac{1}{2}$
$\Delta E_j < 0$	$\Delta E_j > 0$	$\Delta E_j > 0$	$\Delta E_j < 0$
$\mathbf{P}(\Delta E_j < 0) > \frac{1}{2}$		$\mathbf{P}(\Delta E_j > 0) < \frac{1}{2}$	

Table 3.1: Possible scenarios of a single hidden unit energy value change during RBM state changes

The energy change depends on two terms, the first is a difference of hidden units in the current and the next steps, the second is an activation function argument. For those, it is possible to analyse the energy change possibilities for all possible scenarios as shown in Table 3.1 (except for  $h_j^{t+1} = h_j^t$  when energy does not change, so it is just omitted). The energy may increase during the state updates, but the possibility of that is lower than the possibility of decreasing the energy and these events are complementary. That is how RBMs work, they tend to decrease the energy, however increasing its value is still possible because of stochastic dynamics.

The above-mentioned energy changes can be demonstrated by a simple experiment that compares how it behaves during the unit's updates when the stochastic algorithm is used and when it is not (unit is ON when  $P \geq \frac{1}{2}$ ). The results of such an experiment are shown in Figure 3.4, it is observed that when stochastic dynamics is enabled the energy increases in some steps, but in an average sense, it decreases. When the stochastic dynamics is disabled the energy never increases, but it quickly reaches a local minimum that cannot be left over.

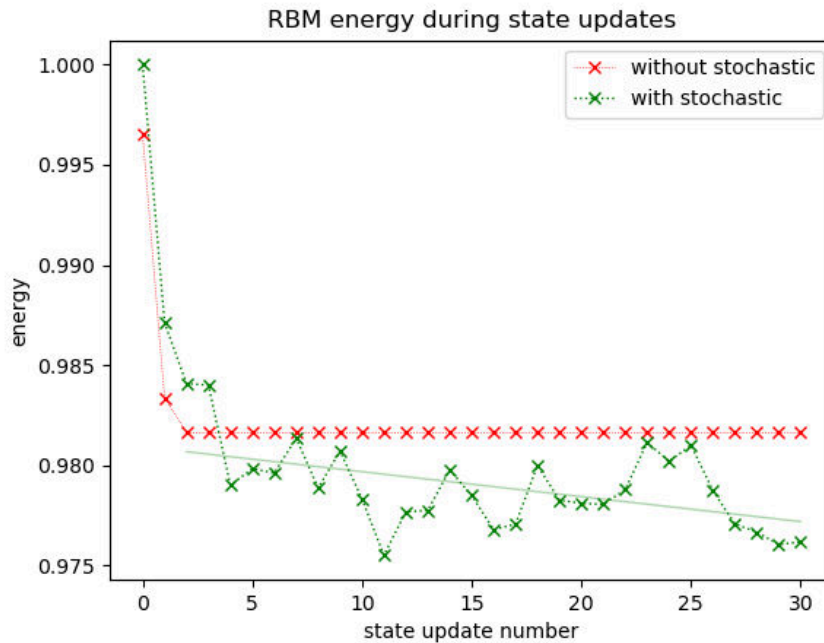


Figure 3.4: Energy value of a Restricted Boltzmann Machine during its state's updates, normalised to its maximum value.

## 3.2 TRAINING AN RBM

As explained, the RBM is a model that learns how to represent a given training dataset -  $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_l\}$  with an unknown probability distribution -  $q$  in some other (latent) space. The learning processes maximises the likelihood over  $\mathcal{S}$ , which is also minimising the Kullback-Leibler Divergence, defined for two distributions  $p$  - model distribution and  $q$  - data distribution

in the following way:

$$D_{\text{KL}}(p \parallel q) = \sum_{\mathbf{s}_i \in \mathcal{S}} p(\mathbf{s}_i) \log \left( \frac{p(\mathbf{s}_i)}{q(\mathbf{s}_i)} \right). \quad (3.22)$$

Based on this formula, RBM training can be approached generally as an MRF. From a practical point of view, a learning algorithm is a set of rules that adjusts a set of parameters, denoted here in a general case as  $\boldsymbol{\theta}$ , and during this process, the values of  $\boldsymbol{\theta}$  should tend to minimise the distance between the  $q$  distribution and  $p$  for a given  $\boldsymbol{\theta}$ . The likelihood -  $\mathcal{L}$  over  $\mathcal{S}$  dataset for the given  $\boldsymbol{\theta}$  can be formulated as follows:

$$\mathcal{L}(\boldsymbol{\theta}^t \mid \mathcal{S}) = \prod_i P(\mathbf{s}_i \mid \boldsymbol{\theta}). \quad (3.23)$$

Maximising the likelihood is the same as maximising the log-likelihood [133] which is more convenient to use as it leads to replacing the product of probabilities with the sum:

$$\log \mathcal{L}(\boldsymbol{\theta}^t \mid \mathcal{S}) = \log \prod_i P(\mathbf{s}_i \mid \boldsymbol{\theta}) = \sum_i \log P(\mathbf{s}_i \mid \boldsymbol{\theta}). \quad (3.24)$$

Just for clarification, this is possible because the  $\log(x)$  is a monotonically increasing function, thus the log-likelihood is monotonically the same as likelihood:

$$P(x \mid \boldsymbol{\theta}_1) > P(x \mid \boldsymbol{\theta}_2) \Leftrightarrow \log P(x \mid \boldsymbol{\theta}_1) > \log P(x \mid \boldsymbol{\theta}_2). \quad (3.25)$$

Training these types of models is relatively complicated. The first techniques were described in [137], the authors there gave a good theoretical background of how Boltzmann Machines should be trained, however, the practical implementation was problematic [133].

Since finding the optimal  $\boldsymbol{\theta}$  is not possible analytically it has to be performed by an iterative method given as  $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \Delta\boldsymbol{\theta}^t$ , which was introduced for MLP in (1.8). It will not ensure that  $\boldsymbol{\theta}$  will be optimal for a given  $\mathcal{S}$ , but the training algorithm should lead to finding parameters for which the model represented by the neural network is sufficient for a certain task. The goal is de facto to find the  $\Delta\boldsymbol{\theta}^t$ , and for a model maximising the likelihood of (3.23), it can be done with a gradient ascent method. In particular, one can write the optimisation formula as follows:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \underbrace{\eta \frac{\partial(\log \mathcal{L}(\boldsymbol{\theta}^t \mid \mathcal{S}))}{\partial \boldsymbol{\theta}^t}}_{\text{gradient ascent term}} - \underbrace{\lambda \frac{\|\boldsymbol{\theta}^t\|^2}{2}}_{\text{stability term}} + \underbrace{\mu \Delta\boldsymbol{\theta}^{t-1}}_{\text{momentum term}}. \quad (3.26)$$

$\underbrace{\hspace{15em}}_{\Delta\boldsymbol{\theta}^t}$

This formula can be considered as an identification/adaptation rule, similar as in identification, control theory, optimisation. The gradient ascent term is related to the maximising likelihood (ML) - and is an essential part of the algorithm,  $\eta$  is a learning rate. The other two terms are used to make the training process more effective. The weight decay term is used to penalise large weights, and  $\lambda$  is a weight decay coefficient. The momentum term is used to accelerate the learning process, thus the  $\Delta\boldsymbol{\theta}^{t-1}$  is sometimes called *velocity* because it increases as the weight change increases, so having this term in an optimisation formula allows to decrease a number of iterations (however the acceleration may depend on the shape of an objective function [136]),  $\mu$  is a momentum coefficient. These parameters affect the stability of a training process.

Formal proofs for some of the following equations are given in [133], so the derivations are shortened for the sake of brevity.

For a particular case of ML learning which is an RBM,  $\mathcal{S}$  is split into  $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_l\}$  and  $\mathcal{H} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_l\}$  for visible and hidden variables, then:

$$\begin{aligned} \ln \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}) &= \ln P(\mathbf{v} | \boldsymbol{\theta}) = \ln \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \\ &= \ln \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} - \ln \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \end{aligned} \quad (3.27)$$

Hence, the gradient can then be computed with the following equation:

$$\begin{aligned} \frac{\partial(\log \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}))}{\partial \boldsymbol{\theta}} &\stackrel{(3.27)}{=} \frac{\partial}{\partial \boldsymbol{\theta}} \left( \log \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \right) - \frac{\partial}{\partial \boldsymbol{\theta}} \left( \log \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \right) \\ &= \frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} + \frac{1}{\sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}} \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} \\ &= - \sum_{\mathbf{h}} P(\mathbf{h} | \mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} + \sum_{\mathbf{v}, \mathbf{h}} P(\mathbf{h}, \mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} \\ &\because p(\mathbf{h} | \mathbf{h}) = \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{v})} = \frac{\frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})}}{\frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}} = \frac{e^{-E(\mathbf{v}, \mathbf{h})}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}} \end{aligned} \quad (3.28)$$

For an RBM, one needs to define update formulas for  $\mathbf{W}$ ,  $\mathbf{a}$  and  $\mathbf{b}$ , to find the derivative for a particular weight  $w_{ij}$  of any given single training sample



$\mathbf{v}_x \in \mathcal{V}$  the following formula can be used [133]:

$$\begin{aligned} \frac{\partial(\ln \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}_x))}{\partial w_{ij}} &= - \sum_{\mathbf{h}} P(\mathbf{h} | \mathbf{v}_x) \frac{\partial E(\mathbf{v}_x, \mathbf{h})}{\partial w_{ij}} + \sum_{v, \mathbf{h}} P(\mathbf{h}, \mathbf{v}_x) \frac{\partial E(\mathbf{v}_x, \mathbf{h})}{\partial w_{ij}} \\ &= P(h_j = 1 | \mathbf{v}_x) v_{xi} - \sum_v P(\mathbf{v}_x) P(h_j = 1 | \mathbf{v}_x) v_{xi} \end{aligned}$$

For all the training examples from a set  $\mathcal{V}$ :

$$\begin{aligned} &\frac{1}{l} \sum_{\mathbf{v}_x \in \mathcal{V}} \frac{\partial(\ln \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}_x))}{\partial w_{ij}} \\ &= \frac{1}{l} \sum_{\mathbf{v}_x \in \mathcal{V}} [\mathbb{E}_{P(\mathbf{h} | \mathbf{v}_x)}[v_i h_j] - \mathbb{E}_{P(\mathbf{h}, \mathbf{v}_x)}[v_i h_j]] \\ &= \langle v_i h_j \rangle_{P(\mathbf{h} | \mathbf{v}_x) q(\mathbf{v}_x)} - \langle v_i h_j \rangle_{P(\mathbf{h}, \mathbf{v}_x)}, \end{aligned}$$

$q(v)$  denotes the empirical distribution and  $\langle \cdot \rangle$  denotes an average expectation with respect to the given probability distribution.

The above analysis leads to the following rule:

$$\sum_{\mathbf{v}_x \in \mathcal{V}} \frac{\partial(\ln \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}_x))}{\partial w_{ij}} \propto \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \quad (3.29)$$

Applying the partial derivatives on log-likelihood for biases  $a_i$ ,  $b_j$  results in the following formulas:

$$\frac{\partial(\ln \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}_x))}{\partial a_i} = v_i - \sum_v P(\mathbf{v}_x) v_i \quad (3.30)$$

$$\frac{\partial(\ln \mathcal{L}(\boldsymbol{\theta} | \mathbf{v}_x))}{\partial b_j} = P(h_j = 1 | \mathbf{v}_x) - \sum_v P(\mathbf{v}_x) P(h_j = 1 | \mathbf{v}_x) \quad (3.31)$$

Finally, having formulated the gradients, one can conclude that computing them is computationally expensive as it involves the  $Z$  constant and the complexity increases exponentially as the  $\boldsymbol{\theta}$  increases so the gradient terms have to be approximated [138]. The approximation may be obtained by running a Gibbs sampling to obtain the model distribution, however, it is still a complex problem because it requires running a Markov chain for a large number of iterations to ensure convergence. As a result, an efficient training algorithm can not be yielded this way. That is why RBMs were not very useful until 2002 when Hinton formulated an interesting simplification of approximation of gradient, and called it Contrastive Divergence (CD) [139]. CD

was de facto invented for training product of experts [140] but an RBM can be regarded as a product of experts [133] so CD is also applicable RBMs. To explain briefly a mathematical motivation, the CD learning approximately follows the following gradient [138]:

$$CD_k \stackrel{(3.22)}{=} D_{KL}(p_0 \parallel q) - D_{KL}(p_k \parallel q), \quad (3.32)$$

where  $k$  is a small number of Gibbs steps used to approximate the gradient. Hinton showed that  $k = 1$  is sufficient to ensure effective learning, hence the needed statistics can be obtained after running one step to compute the first reconstruction, and the last needed equation can be formulated in the following way:

$$\frac{\partial(\ln \mathcal{L}(\boldsymbol{\theta} \mid \mathbf{v}_x))}{\partial w_{ij}} \approx \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{reconstruction} \quad (3.33)$$

This approach significantly reduces the complexity of the learning model because one needs only to compute the first reconstruction and all of the needed statistics can be obtained in parallel since computing the  $\Delta\boldsymbol{\theta}_i$  is independent for every  $i$ . Many practical tips of how to apply CD, and how to train an RBM including choosing good  $\eta$ ,  $\lambda$ ,  $\mu$ , and initialising  $\boldsymbol{\theta}$  are given in [136].

Despite the high speed of CD, it may still have some practical difficulties in terms of memory usage. The default approach is to make an update of weights after all the samples from a training dataset have been processed and the statistics from them have been obtained. If one has a lot of training examples (which is a frequent case for unsupervised learning) the memory needed to store all of them may be large, therefore *batch learning* [141] may be used. For this type of training, the set  $\mathcal{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_l]$  is split into a number of batches  $\mathcal{V} = [\mathcal{V}_1 \ \mathcal{V}_2 \ \dots \ \mathcal{V}_{bn}]$ , where each batch contains a number of training samples  $\mathcal{V}_x = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_{bs}]$ , so  $bn$  denotes the number of batches, and  $bs$  denotes size of a single batch (the last batch may be smaller if  $l$  is not divisible by  $bn$ ). Then an update of the parameters is performed after processing each  $\mathcal{V}_x$ . All the information presented here is presented as in Algorithm 2.

It is also possible to train an RBM in a fully supervised manner. Then formally, the input  $\mathbf{v}$  becomes de facto a vector containing concatenated input and output.

Let  $\mathcal{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_l]$  be a set of input vectors, and  $\mathcal{Y} = [\mathbf{y}_1 \ \mathbf{y}_2 \ \dots \ \mathbf{y}_l]$  be a vector of one-hot encoded labels, corresponding to the  $\mathcal{X}$  set. Then the training set  $\mathcal{V} = [[\mathbf{x}_1 \ \mathbf{y}_1] \ [\mathbf{x}_2 \ \mathbf{y}_2] \ \dots \ [\mathbf{x}_l \ \mathbf{y}_l]]$ . In the prediction phase, having the input  $\mathbf{x}$ , one needs to compose  $\mathbf{v}$  by concatenation  $\mathbf{x}$  with a vector of the

same length as it was for  $\mathbf{y}$  but containing all zeros. Then the one Gibbs step has to be run, and the reconstructed  $\mathbf{v}$  should contain the predicted label at the place of  $\mathbf{y}$ .

---

**Algorithm 2** RBM training with Contrastive Divergence,  $N_e$  denotes the number of training epochs

---

```

epoch  $\leftarrow$  0;
 $\mathbf{W} \leftarrow X_{\sim \mathcal{N}}^{R_h \times R_v} (\mu = 0, \sigma^2 = 0.1)$ ;
 $\mathbf{a} \leftarrow [0, 0, \dots, 0]$ ;
 $\mathbf{b} \leftarrow [0, 0, \dots, 0]$ ;
 $\mathbf{W}_{\text{velocity}} \leftarrow 0 \cdot \mathbf{W}$ ;
 $\mathbf{a}_{\text{velocity}} \leftarrow \mathbf{a}$ ;
 $\mathbf{b}_{\text{velocity}} \leftarrow \mathbf{b}$ ;
while epoch  $<$   $N_e$  do
  batch  $\leftarrow$  1;
  while batch  $\leq$   $bn$  do
     $\mathbf{v} \leftarrow X \sim \mathcal{V}_{\text{batch}}$ ;
     $\mathbf{p}_h \leftarrow \sigma(\mathbf{W}\mathbf{v} + \mathbf{b})$ ;
     $\mathbf{pw}_{\text{stats}} \leftarrow (\mathbf{v}^\top \mathbf{p}_h) / bs$ ;
     $\mathbf{ph}_{\text{stats}} \leftarrow \frac{1}{bs} \sum_{i=1}^{bs} ph_i$ ;
     $\mathbf{pv}_{\text{stats}} \leftarrow \frac{1}{bs} \sum_{i=1}^{bs} v_i$ ;
    step  $\leftarrow$  0;
    while step  $<$   $k$  do
       $\mathbf{h} \leftarrow \mathbf{p}_h > X_{\sim \mathcal{U}}^{R_h}$ ;
       $\mathbf{p}_v \leftarrow \sigma(\mathbf{W}^\top \mathbf{h} + \mathbf{a})$ ;
       $\mathbf{v} \leftarrow \mathbf{p}_v > X_{\sim \mathcal{U}}^{R_v}$ ;
       $\mathbf{p}_h \leftarrow \sigma(\mathbf{W}\mathbf{v} + \mathbf{b})$ ;
      step  $\leftarrow$  step + 1;
    end while
     $\mathbf{nw}_{\text{stats}} \leftarrow (\mathbf{v}^\top \mathbf{p}_h) / bs$ ;
     $\mathbf{nh}_{\text{stats}} \leftarrow \frac{1}{bs} \sum_{i=1}^{bs} ph_i$ ;
     $\mathbf{nv}_{\text{stats}} \leftarrow \frac{1}{bs} \sum_{i=1}^{bs} v_i$ ;
     $\mathbf{W}_{\text{velocity}} \leftarrow \mu \cdot \mathbf{W}_{\text{velocity}} + \eta \cdot (\mathbf{pw}_{\text{stats}} - \mathbf{nw}_{\text{stats}}) - \lambda \frac{\|\mathbf{W}\|^2}{2}$ ;
     $\mathbf{a}_{\text{velocity}} \leftarrow \mu \cdot \mathbf{a}_{\text{velocity}} + \eta \cdot (\mathbf{pv}_{\text{stats}} - \mathbf{nv}_{\text{stats}})$ ;
     $\mathbf{b}_{\text{velocity}} \leftarrow \mu \cdot \mathbf{b}_{\text{velocity}} + \eta \cdot (\mathbf{ph}_{\text{stats}} - \mathbf{nh}_{\text{stats}})$ ;
     $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{W}_{\text{velocity}}$ ;
     $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{a}_{\text{velocity}}$ ;
     $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{b}_{\text{velocity}}$ ;
    batch  $\leftarrow$  batch + 1;
  end while
  epoch  $\leftarrow$  epoch + 1;
end while

```

---

### 3.3 DEEP BOLTZMANN MACHINES

Another interesting property of Restricted Boltzmann Machines is that they can be stacked, which conceptually means the same as adding more layers to an MLP, so having more layers results in the possibility of representing the data in a more abstract manner. There are two types of models that contain RBMs but have more than two layers. The first is a Deep Belief Network (DBN) [142], the second is a Deep Boltzmann Machine (DBM) [143]. They share similar ideas of learning, however differ in the connection types between the layers.

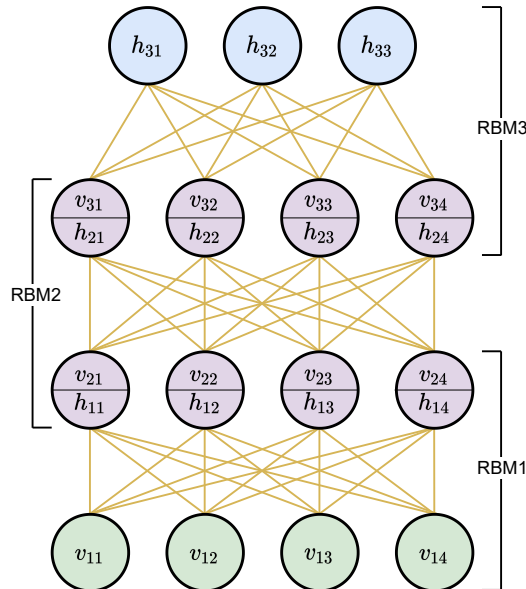


Figure 3.5: Stacking multiple RBMs, DBN/DBM model.

Figure 3.5 presents how the RBMs are stacked to form a DBN or DBM. The topology for both is rather simple, the signal propagates from the very bottom RBM to the higher ones, the key is that the output from one RBM ( $\mathbf{h}$  in this case) becomes an input to the next RBM -  $\mathbf{v}$ . That is why the middle layers in Figure 3.5 have two symbols for their units ( $v_{ij}$  and  $h_{kl}$ ). The obvious question is how these models can be trained, and the answer is surprisingly simple. They can be trained in a greedy layer-wise way, which means the learning starts from RBM1 in an unsupervised manner (Contrastive Divergence), then after the first machine is trained its parameters are no longer changed. Having RBM1 trained, one can commence training RBM2 which is done with the use signal propagated by RBM1. The process

is repeated until the last RBM is trained. This type of training is sometimes described as wake-sleep algorithm, and unsurprisingly has been invented by Geoffrey Hinton [144], and some improvements to the training algorithm of DBM has been proposed in [145]. The last RBM can be also trained with supervision by the inclusion of labels to its visible layer (this was explained in the training section).

The difference between a DBN and a DBM is that in a DBN only the top two layers form an RBM which is an undirected model, whereas the lower layers form a directed generative model, in DBM all the connections are undirected. Therefore DBM may propagate the signal with top-down feedback, that may be useful to deal with an ambiguous input [143].

Despite that DBNs are not commonly used, their importance in AI and impact on image processing cannot be overrated. In short, before DBNs have introduced, deep neural networks were very hard to train because of previously mentioned vanishing gradient problem. DBNs do not have that problem because of the use of different training rules, so the level of abstraction might be much higher. This is important not only because of the use of DBN themselves, but taking into account that RBMs can be reinterpreted as a two-layer MLP, the entire DBN can be reinterpreted as a multilayer MLP. So, for many years RBM training was used to initialise weights of multilayer neural networks, that could then be fine-tuned with the use of backpropagation without factoring-in the vanishing gradient problem. They were successfully applied in many machine learning problems, some of the most improvement of such can be found in [146, 147, 148]. This method of pretraining deep neural networks has since been superseded, because as recent research has shown, the use of ReLU and dropout [149] is sufficient to train complex models by backpropagation only [150].

---

## IMAGE PROCESSING AND CLASSIFICATION WITH BINARY DESCRIPTORS AND RESTRICTED BOLTZMANN MACHINES

---

The previous chapters introduced the basic concepts of neural networks and image processing in terms of classification. They also raised some difficulties related to them. As tools that may potentially mitigate those difficulties, local binary descriptors and an RBM have been shown as a method that can learn in an unsupervised manner. Before revealing these techniques it is worth explaining the conceptual background. What is a binary descriptor? It can be said this is a representation of some unique feature in a binary vector space. No matter what the size of this space is, the significance is that each bit in this space has a special meaning, for example, some bits may denote a bright left corner while other may denote a dark dot pattern, so combining all this information one can form an abstract description of a processed feature. Then, going to their usability for classification, one can say that a specific number of given features in some region of an image may point to some specific category that the processed image belongs to. This is due to the fact that a description of an object is easier to understand by machine than a raw pixel representation of an image.

The data representation in some abstract spaces has been described when an MLP has been presented, so the conclusion is that providing more complex data to a classifier leads to a better performance because it is similar to adding more hidden layers, which would have to be trained in a supervised manner, thus their capacity would be larger resulting in more data being needed to train them effectively. Nevertheless, processing the descriptors is not an easy task for machine learning, after a binary descriptor transformation there are a lot of binary descriptors that may be futile in themselves. One obvious approach to preprocess data is to perform dense binary descriptors

and classify the vectors directly by a CNN. However, such a simple method may not be effective enough, which can be shown in this thesis. Instead, the author of this research proposes a pipeline where the descriptors are followed by a layer of RBMs.

It has been explained what kind of neural network an RBM is, essentially, it can learn a distribution of descriptors given in a training dataset. Hence the latent space of an RBM provides even more abstract information than the binary description layer itself. The hidden space is not necessarily easily understood by humans, but one may correctly assume that it should be easier for a machine, as a neural network uses it to reproduce the input or even generate new samples. Other abilities of an RBM are important for this case too. First, they can be used for an aggregation of the descriptors if their dimensionality varies. Second, they are able to reconstruct the input data to be similar to the training examples. Third, the ability to compute the probability of an occurrence of a given descriptor in the training dataset may result in the possibility of providing a special metrics denoting a similarity between given visual data (image or set of images) and the reference data (training data).

## 4.1 THE RBM AND THE CLBP - AUTHOR'S IMPLEMENTATION

---

At present, Restricted Boltzmann Machines are not frequently applied to common machine learning problems, thus most of the modern libraries that support neural networks do not have an RBM implementation. Even if they do, like Scikit [151] does, they are not sufficient for the research purposes of this dissertation, due to not supporting GPUs and relatively hard modification of the code inside. Therefore, the implementation of an RBM has been prepared for the purposes of this study. The introduction to the RBM showed that all the computations may be done with basic matrix arithmetic, so it is relatively easy to implement. It is required that the application is portable and works on GPUs and CPUs in case of GPUs not being accessible on some devices. This allows running training on a more powerful machine and inference on a target device.

Python was chosen from among the various programming languages. Python is practically slower than other compilable languages, but the goal is to use the optimised libraries for the array computations, so the speed is not a concern in this context as the crucial operations are fast enough. Also, Python is widely known for its simplicity and is supported by many libraries. As a consequence, it is easy to implement other functionalities like files manage-



ment or image preprocessing. For Python, one of the most commonly known and widely-used libraries for matrix arithmetic is NumPy [152]. Its API is very straightforward, therefore the CPU implementation of an RBM is written with the use of this library. The GPU implementation is supported by TensorFlow [153]. It is de facto a large library for many AI purposes, but from version 2.4 it also supports some subset of NumPy operations (API is called TensorFlow NumPy), thus the basic matrix arithmetic can be easily moved from CPU to GPU. The same functionality should be also achievable with the use of PyTorch [154] or JAX [155]. The only cases that the programmer has to care about are the types promotion and copying between the CPU and GPU memories (it is time consuming so the number of these copying has to be limited). This approach has also another advantage, it allows using directly the data processed by an RBM in other python libraries which is important because as mentioned RBM is used only for preprocessing and its output has to be easily forwarded to other modules.

The final implementation supports the following functionalities:

- batch training with CD,
- saving/loading RBM parameters as binary ".npz" files,
- inference on any device supporting Numpy (CPU) or TensorFlow Numpy (GPU),
- computing energy, free energy, conditional and marginal probability of given input,
- reconstruction of an input.

As introduced previously, this research uses RBMs as binary descriptor processors, especially the CLBP. Since the CLBP is a novelty proposed in this dissertation, there is no public implementation of it, so there was a need to create one. It is written with the use of the Python programming language but with the support of GPUs for the same reason as the implementation of an RBM. However, the application of a CLBP is more complicated since it takes advantage of many non-linear transformations on an image and they have to be performed in parallel to ensure short processing time, and this is not possible with NumPy. Despite this, these requirements may be met with the use of Numba [156], a Python library that enables using parallel computing on GPUs by writing one's own function definitions. The implementation is fully parametrisable so flexible in terms of further experiments and supports the following functionalities:

- configurable input size,
- configurable colour mode (RGB/greyscale),
- configurable stride and kernel size (if descriptors are concatenated),
- multistage processing with RBMs (for DBNs).

More code specific details of the implementations of the CLBP and RBM are descriptors in Apeendix [A](#).

---

## 4.2 THE RBM AS A BINARY DESCRIPTORS PROCESSOR AND AGGREGATOR, CD FEATURE SPACE

---

The motivation for using an RBM for binary descriptors aggregation comes from the concept of Contrastive Divergence. Intuitively, we can treat this gradient approximation as a metrics that describes the distance between the input vector and the trained model. The meaning of this is rather abstract, but it is easy to imagine that if an RBM attempts to reconstruct the input vector to a form similar to that being observable during the training, a CD is a metrics of how close the reconstruction is to the input. Therefore we can treat CD as a type of transformation that provides an abstract space describing all the descriptors.

Understanding this, one can define a formula describing the method. Let  $\mathbf{P} \in \mathbb{A}^{h \times w \times c}$  be an input image,  $\mathbf{P}_{(i,j) \sim l}$  be a subtensor of  $\mathbf{P}$  as explained in (2.27), and  $\mathbf{B} \in \{0, 1\}^{n \times k}$  be a features matrix, where  $n$  is a number of processed features, and  $k$  the length of binary descriptor. The binary description:

$$\mathbf{Z} : \mathbf{P} \mapsto \mathbf{B} \quad (4.1)$$

can describe the image transformation in two ways as explained earlier. The first is a dense method that gathers the features for every pixel in the image, or using a stride  $s_1$  to obtain a grid pattern, in this case, the dimensionality of a grid is  $\lfloor \frac{h}{s_1} \rfloor \times \lfloor \frac{w}{s_1} \rfloor$  and will be referred to as  $\hat{h} \times \hat{w}$ . The dimensionality of the output is then  $n = \hat{h} \cdot \hat{w}$  (padding is omitted for simplicity), and the output is computed as follows:

$$\forall_{i \in \{s_1, 2 \cdot s_1, \dots, h - s_1, h\}, j \in \{s_1, 2 \cdot s_1, \dots, w - s_1, w\}} \mathbf{B}_{x,:} = \zeta(\mathbf{P}_{(i,j) \sim l}), \quad (4.2)$$

where  $x = \lfloor \frac{j}{s_1} \rfloor + w \cdot \lfloor \frac{i}{s_1} \rfloor$ .

The second method uses only specific keypoints to compute the features matrix, so for  $\mathbf{p} = (p_1, p_2, \dots, p_l)$  being a set of keypoints, that  $\forall_i p_i \in \mathbb{N}_+^2$ , the binary features vectors can be computed as follows:

$$\forall_{1 \leq x \leq n} \mathbf{B}_{x,:} = \zeta(\mathbf{P}(\mathbf{p}_x)). \quad (4.3)$$

Both methods generate similar matrices but in the second method, a value of  $n$  varies depending on a number of detected keypoints. The idea of using CD from a RBM in case of variable  $n$  is to compute a CD value after first reconstruction and use it for further classification, the result of this is referred as  $\mathbf{C} \in \mathbb{R}^{R_h \times R_v}$ :

$$\mathbf{C} = \frac{1}{n} \left( \sum_{i=1}^n \gamma_1(\mathbf{B}_{i,:}) \right), \quad (4.4)$$

where  $\gamma_k(\cdot)$  denotes a result of a Contrastive Divergence from a single binary vector after  $k$  steps for reconstruction (for simplicity:  $\gamma(\cdot) \triangleq \gamma_1(\cdot)$ ). To recall the part of the Contrastive Divergence algorithm we can consider the following formula:

---

**Algorithm 3** Contrastive Divergence for an input vector -  $\mathbf{v}$

---

$$\begin{aligned} \mathbf{p}_h &\leftarrow \sigma(\mathbf{W}\mathbf{v} + \mathbf{b}); \\ \mathbf{h} &\leftarrow \mathbf{p}_h \succ X_{\sim \mathcal{U}}^{R_h}; \\ \mathbf{p}_v &\leftarrow \sigma(\mathbf{W}^\top \mathbf{h} + \mathbf{a}); \\ \tilde{\mathbf{v}} &\leftarrow \mathbf{p}_v \succ X_{\sim \mathcal{U}}^{R_v}; \\ \tilde{\mathbf{p}}_h &\leftarrow \sigma(\mathbf{W}\tilde{\mathbf{v}} + \mathbf{b}); \\ \gamma(\mathbf{v}) &\leftarrow \mathbf{v}^\top \mathbf{p}_h - \tilde{\mathbf{v}}^\top \tilde{\mathbf{p}}_h; \end{aligned}$$


---

CD obtained for all descriptors is defined as  $\mathbf{\Gamma}(\cdot)$  as follows:

$$\mathbf{\Gamma}(\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_m]) = [\gamma(\mathbf{v}_1) \ \gamma(\mathbf{v}_2) \ \dots \ \gamma(\mathbf{v}_m)] \quad (4.5)$$

The  $\mathbf{C}$  matrix may be stacked to a vector  $\mathbf{c}$  of size  $R_v \cdot R_h$ . Due to the average one can obtain a vector of constant size even with a variable value of  $n$ , however averaging the features from an entire image may be not efficient for complex input data, so this method may be expanded to describe prior defined regions from an image, then the feature space becomes a matrix:

$$\mathbf{C}^* = [\mathbf{c}_1 \ \mathbf{c}_2 \ \dots \ \mathbf{c}_m], \forall_i \mathbf{c}_i \in \mathbb{R}^{R_h \cdot R_v},$$

where  $m$  is a constant number of selected regions, and each  $\mathbf{c}_i$  vector represents a feature from a given region of an image, then:

$$\mathbf{\Gamma} : \{0, 1\}^{n \times k} \rightarrow \mathbb{R}^{m \times (R_h \cdot R_v)}.$$


---

For the region selection one can employ a Selective Search algorithm [157], which is used to infer regions of interest based on image colour distribution. After all these operations  $\mathbf{C}^*$  may be processed by a final classifier, the approach based on keypoints and averaging results in a feature space of relatively low dimensionality that potentially should be easy to classify. Thus the proposal is to use a  $k$  nearest neighbours procedure (KNN), which is a very fast classification method. A simplified pipeline of the processing in this case is shown in Figure 4.1.

**Architecture 1 (CD-KNN).** Processing data with CD feature space aggregation, Selective Search as region selection, and K nearest neighbors as a final classifier.

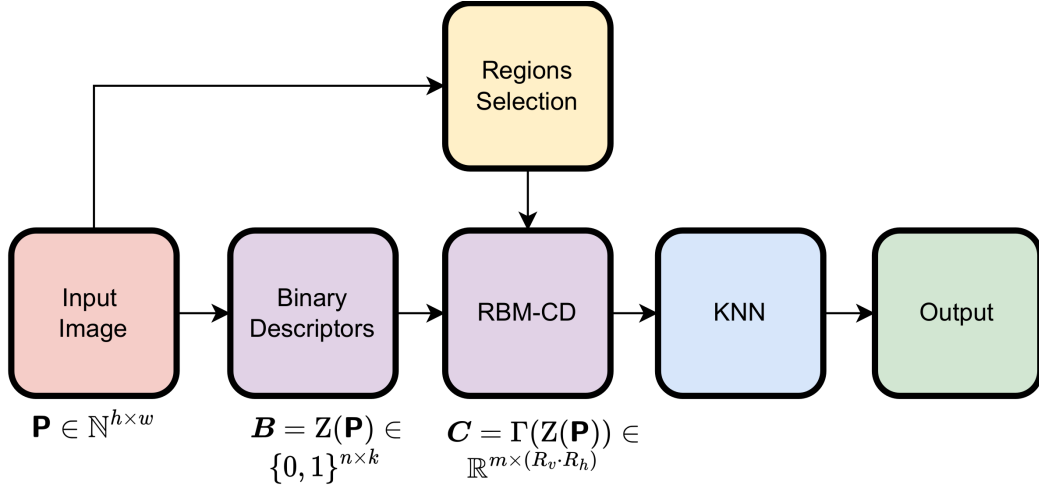


Figure 4.1: Image classification with preprocessing with an RBM and Contrastive Divergence feature space. Binary descriptors are gathered from key-points and CD is averaged for  $m$  prior given regions of an image.

On the other hand, when descriptors are taken from constant points of the image, the features space may not represent regions of interest properly, hence, they have to be inferred by other processing blocks. This is due to the 2D nature of objects being recognised, the transformation of an image to a vector of descriptors may derange the context of the shapes of objects. Therefore, the author of this study proposed another solution which uses a CNN after expansion to a CD space, the CD then becomes a 3D tensor (or vector of matrixes for better understanding its form) of a more abstract meaning:

$$\mathbf{C} = [\mathbf{C}_1 \ \mathbf{C}_2 \ \dots \ \mathbf{C}_{R_h \cdot R_v}], \forall_i \mathbf{C}_i \in \mathbb{R}^{\hat{h} \times \hat{w}}.$$

The  $\mathbf{C}_i$  matrix is obtained as the  $i$ -th element from Contrastive Divergence feature vector. The tensor  $\mathbf{C}$  has the same form as the input to a regular convolution layer, and similar meaning because the composition of feature values is the same as the composition of pixels in a raw image. So we can replace the input to a CNN, which by default is  $\mathbf{P}$ , with  $\mathbf{C} = \Gamma(\mathbf{Z}(\mathbf{P}))$ . A schematic of the entire processing pipeline is shown in Fig 4.2.

**Architecture 2 (CD-CNN).** Processing data with a CD feature space on densely gathered descriptors and a CNN-MLP as a final classifier.

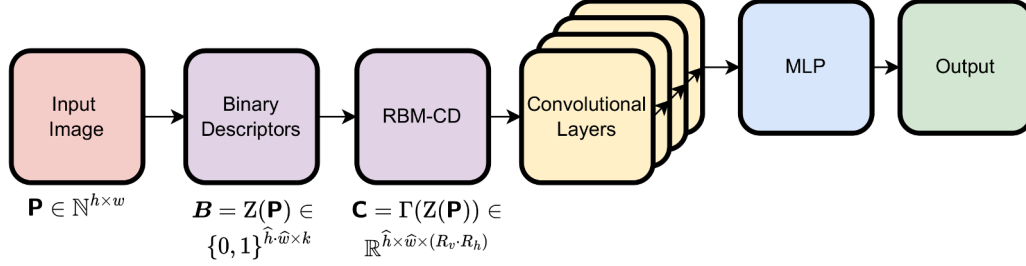


Figure 4.2: Image classification with preprocessing with an RBM and a Contrastive Divergence feature space. Binary descriptors are gathered densely.

This architecture is the author’s contribution and introduces two additional stages of processing when compared to a regular CNN. However, it does not make the entire architecture much more complex since the binary descriptors layer and RBM layer can be made computationally efficient and most of the computing resources are needed for the CNN and MLP. The fact, that this preprocessing does not require training with supervision makes it useful in the case of using unlabelled data for training. The main advantage of this approach is that  $\mathbf{C}$  has a more abstract meaning than  $\mathbf{P}$  so the CNN can process features that are more complex than the raw representation of an image. Usually, these features are detected in further convolutional layers that have to be trained on labelled data, due to the use of the proposed preprocessing the process of supervised learning features detection may be limited by reducing the number of training samples or number of convolutional layers.

### 4.3 THE RBM AS A BINARY DESCRIPTORS PROCESSOR, HIDDEN LAYER FEATURE SPACE

---

The previous section introduced a preprocessing method proposed by the author that enhances an input to a CNN with the use of binary descriptors and a Restricted Boltzmann Machine. The presented feature space was computed based on Contrastive Divergence. This value describes the distance between the input vector and a reconstruction vector given by an RBM which intuitively provides a good abstract space of an image that can be further processed. However, there is also another possibility of computing abstract

features from an image with a similar pipeline. One can view an RBM as an autoencoder and then its hidden layer can be regarded as a complex representation of an input being a binary descriptor. In other words, an RBM can be trained on a set of binary vectors as it was for the CD feature space, but then the structure of an RBM can be reinterpreted as a single-layer feed-forward neural network, in which the values of hidden units can be further processed by a CNN. The motivation is the same as it was in the case of the CD feature space, so the idea of unsupervised training of the preprocessing layer to enrich the input signal remains the same, but the computations are simplified.

First, an input to an RBM layer is a binary description  $\mathbf{B} = \mathbf{Z}(\mathbf{P})$  and all the descriptors are taken densely, thus the dimensionality of  $\mathbf{B}$  is  $\hat{h} \cdot \hat{w} \times k$ , but for further consideration to make its composition similar to an input image,  $\mathbf{B}$  can be reinterpreted as a tensor  $\mathbf{B} \in \{0, 1\}^{\hat{h} \times \hat{w} \times k}$ , in such a case the  $i$ -th bit of a binary descriptor becomes a part of  $i$ -th channel in this tensor. Then, RBMs process  $\mathbf{B}$  channel-wise which means that each binary vector is processed separately by a single RBM, but the weights of the RBM are the same for every part of an image. Figure 4.3 presents a visualisation of this process.

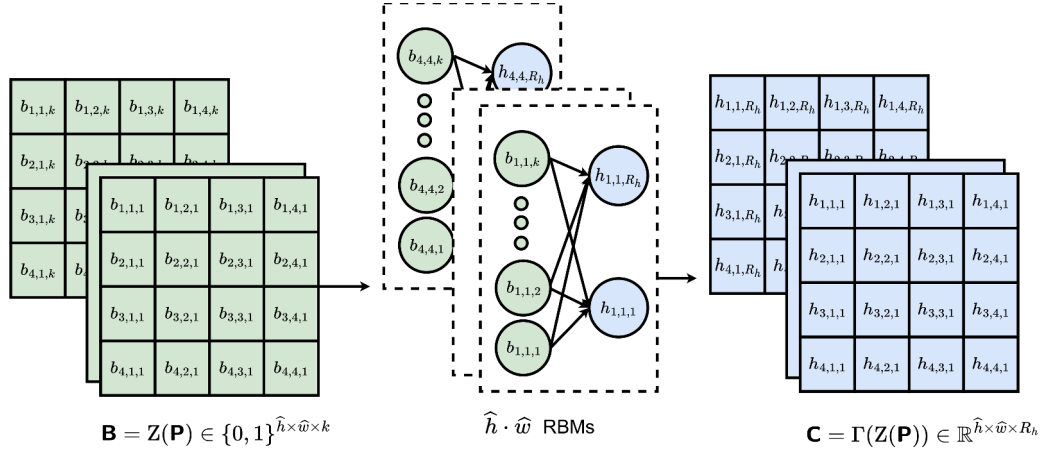


Figure 4.3: Image feature extraction with binary descriptors and RBM hidden space.

In this case,  $\hat{h} \cdot \hat{w}$  matrix multiplications have to be performed to compute the hidden state of RBMs, however, as the weights are shared between RBMs the entire operation can be reinterpreted as a single matrix multiplication performed on  $\mathbf{B}$  matrix, hence:

$$\mathbf{C} = \Gamma(\mathbf{B}) = \varphi(\mathbf{W}\mathbf{B} + \mathbf{b}) \quad (4.6)$$

then  $\mathbf{C}$  of size  $\hat{h} \cdot \hat{w} \times R_h$  has to be reshaped to a  $\hat{h} \times \hat{w} \times R_h$  shape.

**Remark** (An RBM’s output scaling). An important point to note is that the sigmoid activation function of an RBM does not have to be used in (4.6) to define  $\varphi(\cdot)$  since the RBM is reinterpreted as a regular feed-forward network. Its hidden space does not have to represent the probability of its activation as  $[0; 1]$ , it can be scaled with a hyperbolic tangent function to  $[-1; 1]$  or ReLU to  $[0; \infty)$  or even identity to  $(-\infty; \infty)$ . The activation function here has a different meaning than it was in the case of the MLP, as discussed earlier. The weights in the RBM are not changed after they have been trained with the Contrastive Divergence algorithm, so during training a CNN, gradients do not propagate to them, thus choosing  $\varphi(\cdot)$  is more like scaling the input to a CNN. Also, the total input to an activation can be scaled with a  $\beta$  factor:

$$\mathbf{C} = \varphi(\beta \cdot (\mathbf{W}\mathbf{B} + \mathbf{b})), \quad (4.7)$$

which results in a different wider ( $\beta < 1$ ) or narrower ( $\beta > 1$ ) slope for a hyperbolic tangent and sigmoid functions as presented in Fig 4.4;  $\beta$  affects these two functions in a similar way since these are closely related ( $\tanh(x) = 2\sigma(2x) - 1$ ). For ReLU and identity functions this procedure is not necessary since the scaling factor may be achieved in further layers.

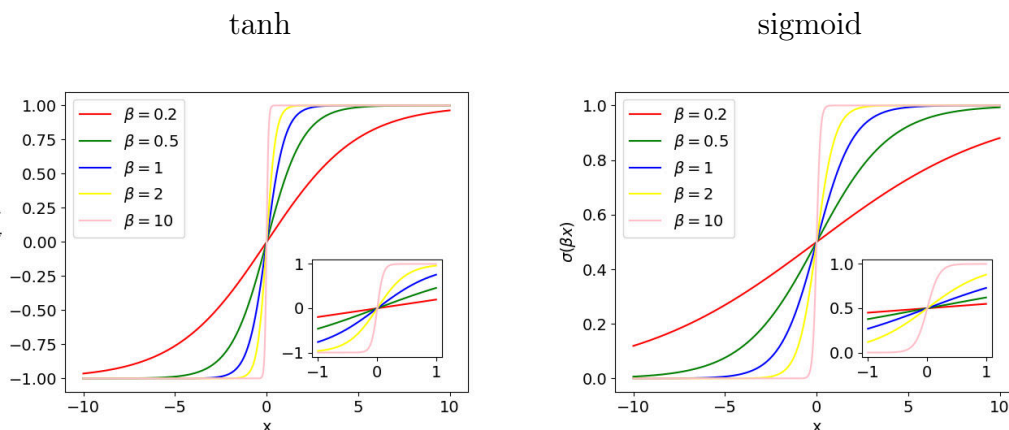


Figure 4.4: Hyperbolic tangent and sigmoid activation functions depending on the  $\beta$  scaling factor.

After the image is transformed and reshaped to the  $\mathbf{B}$  tensor, the input to an RBM layer can be formed in a more complex manner. Binary vectors may be concatenated to create a wider context for an RBM, which may infer more complex features from a binary input. To do this one may use a kernel of size  $k_s$ . A kernel here has a similar meaning to a filter in convolutional layers,



it is designed to take descriptors from some region of an image rather than a single descriptor. Fig 4.5 presents a simple example of how many binary vectors shape an input to an RBM in the case of  $k_s = 2$ . These kernels are obtained densely, however a grid pattern with a stride may also be used, the stride size here will be referred to as  $s_2$ .

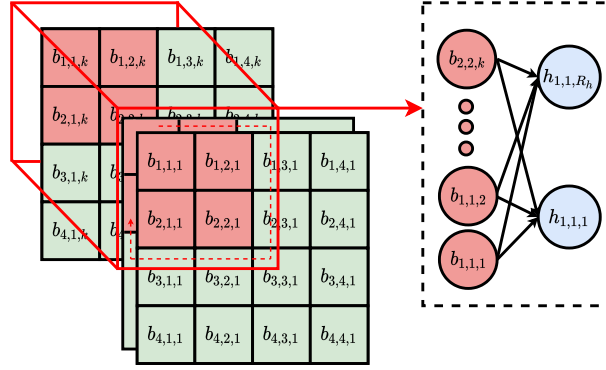


Figure 4.5: Image feature extraction with binary descriptors and RBM hidden space with kernel size equal to 2.

The dimensionality of an input matrix to an RBM layer changes to  $\frac{\hat{h} \cdot \hat{w}}{s_2^2} \times k_s^2 \cdot k$ , as the input vector to a single RBM enhances from  $\{0, 1\}^k$  to  $\{0, 1\}^{k_s^2 \cdot k}$ . The dimensionality of  $\mathbf{C}$  changes to  $\frac{\hat{h}}{s_2} \times \frac{\hat{w}}{s_2} \times R_h$ . The new input, formed from  $\mathbf{B}$  will be referred to as  $\tilde{\mathbf{B}}$ , the transformation will be referred to as  $\mathbf{H}(\cdot)$ , and has the following form:

$$\tilde{\mathbf{B}} = \mathbf{H}(\mathbf{B}) = \begin{bmatrix} \mathbf{B}_{1,1,:} & \cdots & \mathbf{B}_{1,k_s,:} & \cdots & \mathbf{B}_{k_s,1,:} & \cdots & \mathbf{B}_{k_s,k_s,:} \\ \mathbf{B}_{1,1+s_2,:} & \cdots & \mathbf{B}_{1,k_s+s_2,:} & \cdots & \mathbf{B}_{k_s,1+s_2,:} & \cdots & \mathbf{B}_{k_s,k_s+s_2,:} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{B}_{1,\hat{w}-k_s,:} & \cdots & \mathbf{B}_{1,\hat{w},:} & \cdots & \mathbf{B}_{k_s,\hat{w}-k_s,:} & \cdots & \mathbf{B}_{k_s,\hat{w},:} \\ \mathbf{B}_{1+s_2,1,:} & \cdots & \mathbf{B}_{1+s_2,k_s,:} & \cdots & \mathbf{B}_{k_s+s_2,1+s_2,:} & \cdots & \mathbf{B}_{k_s+s_2,k_s+s_2,:} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{B}_{\hat{h}-k_s,\hat{w}-k_s,:} & \cdots & \mathbf{B}_{\hat{h}-k_s,\hat{w},:} & \cdots & \mathbf{B}_{\hat{h},\hat{w}-k_s,:} & \cdots & \mathbf{B}_{\hat{h},\hat{w},:} \end{bmatrix}. \quad (4.8)$$

To exemplify, for  $k_s = 2$  and  $s_2 = 1$  it takes the following form:

$$\tilde{\mathbf{B}} = \begin{bmatrix} \mathbf{B}_{1,1,:} & \mathbf{B}_{1,2,:} & \mathbf{B}_{2,1,:} & \mathbf{B}_{2,2,:} \\ \mathbf{B}_{1,2,:} & \mathbf{B}_{1,3,:} & \mathbf{B}_{2,2,:} & \mathbf{B}_{2,3,:} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}_{1,\hat{w}-1,:} & \mathbf{B}_{1,\hat{w},:} & \mathbf{B}_{2,\hat{w}-1,:} & \mathbf{B}_{2,\hat{w},:} \\ \mathbf{B}_{2,1,:} & \mathbf{B}_{2,2,:} & \mathbf{B}_{3,1,:} & \mathbf{B}_{3,2,:} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}_{\hat{h}-1,\hat{w}-1,:} & \mathbf{B}_{\hat{h}-1,\hat{w},:} & \mathbf{B}_{\hat{h},\hat{w}-1,:} & \mathbf{B}_{\hat{h},\hat{w},:} \end{bmatrix}. \quad (4.9)$$

The RBM layer performs the same computation as earlier:

$$\mathbf{C} = \varphi(\beta \cdot (\mathbf{W}\tilde{\mathbf{B}} + \mathbf{b})).$$

Based on that, one may conclude that the kernels are similar to receptive fields in convolutions. They are designed to process larger regions from an image, therefore the hidden space of RBMs represent a wider context, but their usability may depend on a binary vector space. As the size of the input vector to a single RBM increases by  $k_s^2$  factor, for long descriptors  $k_s > 1$  may result in an inefficient RBM layer, also longer descriptors usually use a larger neighbourhood to compute the output, so the context of a larger region is embedded in a descriptor. Therefore this method is designed for short descriptors like LBP8s or CLBPs. The entire processing pipeline is shown in Figure 4.6

**Architecture 3 (HS-CNN).** Processing data with RBM hidden space (HS) as features and CNN and MLP as a final classifier.

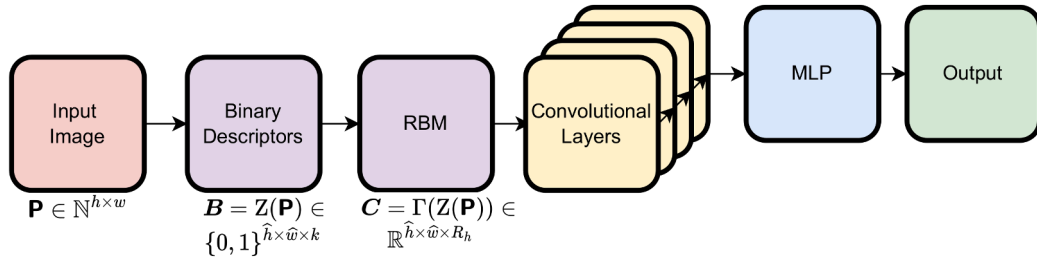


Figure 4.6: Image classification with preprocessing with RBM hidden space as feature extractor. Binary descriptors are gathered densely.

In Section 3.3 the possibility of stacking multiple RBMs was introduced, which results in a higher abstraction level. This can also be applied in the

case of the proposed processing. The idea of using more layers of RBMs is the same as in the case of the DBN, applying this to the processing pipeline changes the  $\Gamma(\cdot)$  function to the following form:

$$\mathbf{\Gamma}(\mathbf{B}) = \mathbf{\Gamma}_r(\mathbf{\Gamma}_{r-1}(\dots(\mathbf{\Gamma}_2(\mathbf{\Gamma}_1(\mathbf{B}))))), \quad (4.10)$$

where  $r$  is the number of RBM layers. However, only  $\mathbf{\Gamma}_r(\cdot)$  can have the form of (4.6) as others ( $\mathbf{\Gamma}_{i \in \{1,2,\dots,r-1\}}$ ) have to perform the regular RBM equations to provide a binary input to the additional layers. As a result, all the RBM layers can be formalised in the following way:

$$\begin{aligned} \mathbf{C}_0 &= \mathbf{B}, \\ \forall_{i \in \{1,2,\dots,r-1\}} \mathbf{C}_i &= \mathbf{\Gamma}_i(\mathbf{C}_{i-1}) = \sigma(\mathbf{W}_i \mathbf{C}_{i-1} + \mathbf{b}_i) > X_{\sim \mathcal{U}}, \\ \mathbf{C} &= \mathbf{\Gamma}(\mathbf{B}) = \mathbf{\Gamma}_r(\mathbf{C}_{r-1}) = \varphi(\beta \cdot (\mathbf{W}_r \mathbf{C}_{r-1} + \mathbf{b}_r)), \end{aligned} \quad (4.11)$$

where  $\mathbf{W}_i$ ,  $\mathbf{b}_i$  denote parameters of an RBM in the  $i$ -th RBM layer, and  $\mathbf{\Gamma}_r(\cdot)$  differs from  $\mathbf{\Gamma}_i(\cdot)$  in activation function as mentioned earlier. This results in the form of a DBN between binary vectors and convolutional layers. However, all the RBM layers may be independent of the previous ones, which means the input can be formed with the use of kernels as it was in the case of forming the input from the binary descriptors layer. So for each RBM one may use the  $\mathbf{H}(\cdot)$  transformation, and finally one can write the following equation:

$$\begin{aligned} \mathbf{C}_0 &= \mathbf{B}, \\ \forall_{i \in \{0,1,\dots,r-1\}} \tilde{\mathbf{C}}_i &= \mathbf{H}_i(\mathbf{C}_i), \\ \forall_{i \in \{1,2,\dots,r-1\}} \mathbf{C}_i &= \mathbf{\Gamma}_i(\tilde{\mathbf{C}}_{i-1}) = \sigma(\mathbf{W}_i \tilde{\mathbf{C}}_{i-1} + \mathbf{b}_i) > X_{\sim \mathcal{U}}, \\ \mathbf{C} &= \mathbf{\Gamma}(\mathbf{B}) = \mathbf{\Gamma}_r(\tilde{\mathbf{C}}_{r-1}) = \varphi(\beta \cdot (\mathbf{W}_r \tilde{\mathbf{C}}_{r-1} + \mathbf{b}_r)). \end{aligned} \quad (4.12)$$

For each layer, there is a different  $\mathbf{H}_i(\cdot)$  since its parameters (stride and kernels size) may differ. Therefore to define the dimensionality of the output tensors and the intermediate matrixes, one needs to define the parameters set. Let  $\mathbf{s} = [s_1 \ s_2 \ \dots \ s_r]$  be a set of strides, and  $\mathbf{k} = [k_1 \ k_2 \ \dots \ k_r]$  be a set of kernel sizes. Then the dimensionality of the output tensor  $\mathbf{C}$  is:

$$\frac{\hat{h}}{\prod_{i=1}^r s_i} \times \frac{\hat{h}}{\prod_{i=1}^r s_i} \times R_{h_r},$$

and it is independent of  $\mathbf{k}$ , but the intermediate matrixes are, for the  $x$ -th RBMs the output dimension is defined in the following way:

$$\frac{\hat{h} \cdot \hat{w}}{\prod_{i=1}^x s_i^2} \times k_x^2 \cdot R_{h_{x-1}}.$$

A simplified visualisation of the entire process is shown in Figure 4.7.

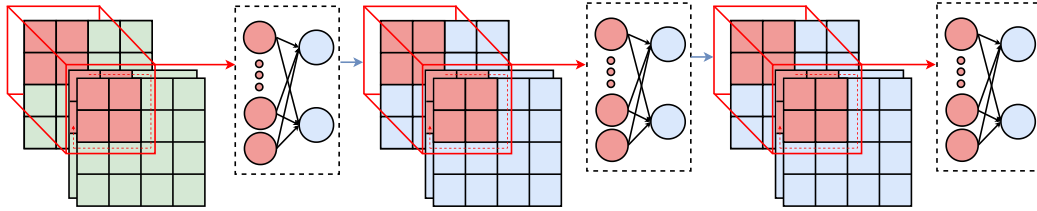


Figure 4.7: Image feature extraction with binary descriptors and multiple RBM layers.

It cannot be predefined what the number of RBM layers should be and what is the best set of  $\mathbf{k}$ ,  $\mathbf{s}$  parameters. However, the visual data is processed similarly to when convolutional layers transform the raw pixels. This means that each layer should have a higher level of abstraction and may detect the features from a wider context. Thus the meaning of kernel size and stride is close to these parameters in convolutions. The idea of stacking RBM layers in this way to create a feature extractor is also known and was introduced in [158], however binary descriptors were not used there (real-valued - Gaussian visible units were used in the first layer). It will be shown later that adding this layer improves the generalisation ability of a convolutional neural network, also the following sections describe additional advantages of using the proposed type of preprocessing.

## 4.4 RBMS FOR VISUAL DATA COMPARISON

The previous sections in this chapter introduced how binary descriptors and Restricted Boltzmann Machines can be utilised as a feature extractor. The preprocessing method presented needs a training phase for an RBM, this is not very demanding as the training examples do not have to be labelled, but it is still a process that has to be performed offline and needs some time and data. Here, arises the question of whether an RBM that has been trained on a dataset can be transformed into another machine learning problem. The idea of transfer learning [62] is commonly known and widely used, and one can claim that this is an example of reusing a feature extractor. However, to make it usable one needs to ensure that the data that is used for classification is similar to the data that has been used for training. The similarity is hard to define, but we can treat it intuitively: feature extractor layers may be reused if the features being detected in the training and classification dataset are related, so if the distributions of particular features are similar

then the extractor should work well. There is no common measure that can be utilised for this purpose, however, for large scale recognition it is probably not necessary. For example, feature extractor learnt on ImageNet are versatile since this dataset has a large number of different objects containing many different features. Thus, reusing a part of classifier trained on ImageNet for other tasks should result in correctly detected features because they were probably similarly distributed in training (but there may be some redundancy since ImageNet may be a more demanding task than others). However, this is still the case when the data that is being recognised presents objects in a similar context such as animals, vehicles, people, buildings etc. We should not expect a good classification performance based on ImageNet for example in texture recognition, hence the problem with measuring data similarity arises again. This leads to the conclusion that despite transfer learning being a very efficient tool to mitigate the difficulties with a lack of data or lack of computing resources (to train big models) it is not clear for a given task if the feature extractor can be moved to another problem, especially for smaller networks and datasets. In contrast to an RBM, regular convolutional neural networks do not provide anything that could indicate if the image features are detected correctly without analysing their output by an expert.

A Restricted Boltzmann Machine may be used to compute the marginal probability of input data as given in (3.17). The  $P(\mathbf{v})$  value defines the probability of occurrence of a given  $\mathbf{v}$  in the dataset. Therefore, with the use of many input signals from a dataset one can compute a vector of their probabilities and compare this to other one (reference vector of probabilities). To formalise it, let  $\mathbf{\Lambda}(\cdot)$  be a transformation that computes a vector of probabilities ( $\mathbf{l}$ ) from a given dataset -  $\mathcal{V}$ :

$$\begin{aligned} \mathbf{l} \triangleq \mathbf{\Lambda}(\mathcal{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_n]) &= [P(\mathbf{v}_1) \ P(\mathbf{v}_2) \ \dots \ P(\mathbf{v}_n)] \in \mathbb{R}^n \\ \forall_{i \in \{1, 2, \dots, n\}} \mathbf{v}_i &\in \{0, 1\}^{R_v}. \end{aligned} \tag{4.13}$$

With the use of  $\mathbf{l}$  one may create a histogram of these probabilities. Due to the real-valued nature of probability, the histogram  $\mathbf{r} \triangleq \mathbf{M}(\mathbf{l})$  can be binned to have a constant size -  $r_s$  ( $\mathbf{r} \in \mathbb{N}^{r_s}$ ). To perform the binning, bin margins have to be defined  $\mathbf{m} = [m_0 \ m_1 \ \dots \ m_{r_s-1} \ m_{r_s}]$ , where  $m_0 = 0$  and  $m_{r_s} = \max(\mathbf{l})$ , then the histogram can be created with the use of the following formula:

$$\forall_{i \in \{1, 2, \dots, r_s\}} r_i = \sum_{x=1}^n ((l_x > m_{i-1}) \wedge (l_x < m_i)). \tag{4.14}$$

The number of bins has to be defined by the user; one way to do this is to use Scott's Rule [159]. Then to obtain the most important part of the histogram one may use an Interquartile Range (IQR). The default values for  $\mathbf{m}$  vectors is to compute them in this way:  $\forall_{i \in \{1, 2, \dots, r_s\}} m_i = \frac{i}{r_s} \cdot \max(\mathbf{l})$ . However, in the case of  $P(\mathbf{v})$  this may result in a large disproportionality, for example most of the probabilities may be accumulated in one bin while others may be empty. To avoid this, the histogram may be flattened to have same height of bins as presented in 4.8.

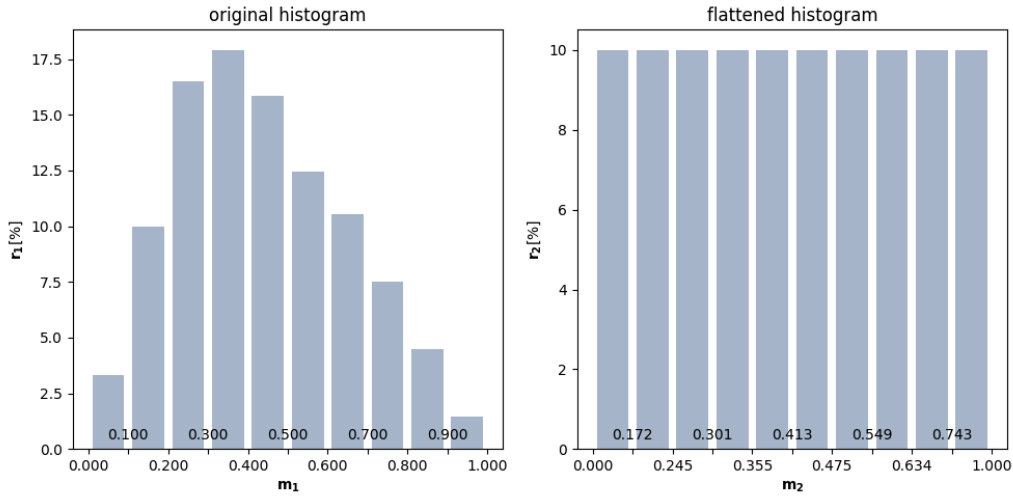


Figure 4.8: Flattening a histogram by adjusting an  $\mathbf{m}$  vector (compare various x-scales in both figures).

Assuming an RBM has been already trained on a training dataset  $\mathcal{X}_{\text{train}}$ , one may extract a smaller subset -  $\mathcal{Z}_{\text{ref}}$  being its representative subset (random samples from different categories). The shapes of  $\mathbf{M}(\mathbf{\Lambda}(\mathcal{X}_{\text{train}}))$  and  $\mathbf{M}(\mathbf{\Lambda}(\mathcal{Z}_{\text{ref}}))$  should be similar because the samples from both contain similar features, hence similar binary vectors.

Then  $\mathbf{r}_{\text{ref}} = \mathbf{M}(\mathbf{\Lambda}(\mathcal{Z}_{\text{ref}}))$  is a reference histogram for a given RBM and to compare new data  $\mathcal{X}_{\text{test}}$  with  $\mathcal{X}_{\text{ref}}$ , one has to create a histogram in the same way. A subset  $\mathcal{Z}_{\text{test}}$  can be also used if  $\mathcal{X}_{\text{test}}$  is large. After that, data comparison is just a comparison of  $\mathbf{r}_{\text{test}} = \mathbf{M}(\mathbf{\Lambda}(\mathcal{Z}_{\text{test}}))$  and  $\mathbf{r}_{\text{ref}}$ . To compute  $\mathbf{r}_{\text{test}}$  one has to use the same  $\mathbf{m}$  vector that has been used for the reference histogram, which implies that although the procedure is quite computationally expensive it does not cause any difficulties as it has to be computed only once for a given RBM. Finally,  $\mathbf{r}_{\text{ref}}$  and  $\mathbf{m}$  are the parameters that describe the data that an RBM has been trained on.

The problematic part of this algorithm may be a partition function (defined in (3.11), in which an exact value of  $Z$  is impossible to compute for high dimensional RBMs. However, this can be mitigated by modifying  $\Lambda(\cdot)$  function to have the following form:

$$\Lambda(\mathcal{V}) = [Z \cdot P(\mathbf{v}_1) \ Z \cdot P(\mathbf{v}_2) \ \dots \ Z \cdot P(\mathbf{v}_n)], \quad (4.15)$$

and taking the marginal probability function into account one can write:

$$\Lambda(\mathcal{V}) = [e^{-F(\mathbf{v}_1)} \ e^{-F(\mathbf{v}_2)} \ \dots \ e^{-F(\mathbf{v}_n)}], \quad (4.16)$$

where  $F(\cdot)$  refers to a free energy function as shown in (3.16). Thus, finally  $Z$  is not necessary as its value is the same for the given parameters of an RBM ( $\mathbf{W}$ ,  $\mathbf{a}$ ,  $\mathbf{b}$ ). As a result, after the parameters have been optimised, the partition function value does not change. However, as  $Z \cdot P(\mathbf{v})$  may be very large, for practical reasons to avoid overflow one may have to use a scaling factor in  $\Lambda(\cdot)$  to have the following form:

$$\lambda(\mathbf{v}) \triangleq \frac{e^{-F(\mathbf{v})}}{G},$$

$$\Lambda(\mathcal{V}) = [\lambda(\mathbf{v}_1) \ \lambda(\mathbf{v}_2) \ \dots \ \lambda(\mathbf{v}_n)], \quad (4.17)$$

where  $G$  is a constant large enough. After the definition of all the computations needed for  $\mathbf{r}_{\text{ref}}$  and  $\mathbf{r}_{\text{test}}$  there is a need to define a function that measures the similarity of these histograms. A standard approach is to use Chi-Squared ( $\chi^2$ ) distance [180] given as:

$$\chi^2(\mathbf{p}, \mathbf{q}) = \frac{1}{2} \sum_{i=1}^{r_s} \frac{(p_i - q_i)^2}{(p_i + q_i)}. \quad (4.18)$$

This formula will be used in further experiments, however, other equations that measure the distance between two vectors could be also employed.

---

## 4.5 RBMS FOR INPUT DATA DENOISING

---

Images being processed by classification systems may come from different sources and can be gathered from different environmental conditions, thus they may be often noised or broken. The noise may have variety of forms, which is not always known prior and denoising the image without knowing the type of noise may be difficult. A theoretical form of the image data distributed by noise can be shown as follows:

$$\tilde{\mathbf{P}} = \mathbf{P} + \mathbf{Y}(\mathbf{P}) + \mathbf{N},$$

where  $\mathbf{P}$  is an original image,  $\tilde{\mathbf{P}}$  is a noised image,  $\mathbf{Y}(\cdot)$  is a function that generates a noise depending on the original image, and  $\mathbf{N}$  is a bias. In real systems  $\tilde{\mathbf{P}}$  is a correlated noise that may come from a processing pipeline, and  $\tilde{\mathbf{N}}$  may come from partially broken sensors or electronic noise or unusual lighting conditions. A good review of existing techniques that tackle denoising problems in both cases is given in [160]. The authors described many methods, including neural ones, and stated that CNN based models achieve the best quality of denoising.

This dissertation focuses on data classification quality, so the form of an original image before being noised is not necessarily needed to be known, the only thing one wants to achieve in this case is to reduce the impact of noise on a classification result, which is important because as stated in [161] even simple noise may affect overall network accuracy. It would be reasonable to reconstruct the original image before passing it to the classification pipeline, however, this results in an additional stage of processing and this is not desirable especially in real-time systems, since the noise nature is not known prior and complex denoising methods are relatively resources consuming. Therefore, as typical denoising methods focus on computing a tensor being a close form of  $\mathbf{P}$ , for classification purposes in the case of RBM preprocessing, the pipeline needs to generate  $\mathbf{\Gamma}(\mathbf{Z}(\tilde{\mathbf{P}}))$  as close as possible to  $\mathbf{\Gamma}(\mathbf{Z}(\mathbf{P}))$ . Finally, the conclusion is that there is no need to reconstruct the original image, and this problem is beyond the scope of this study.

It is not possible to change the  $\mathbf{Z}(\cdot)$  function since binary descriptors are pre-defined and they do not modify the noised input to be similar to the original, however some features of binary descriptors may be successfully applied to tackle denoising [162] as they may generate the same output that differ to a small degree, so the first stage of reducing the impact of noise is applied in this stage. Then the core of the denoising can be implemented by modifying the  $\mathbf{\Gamma}(\cdot)$  function in order to reconstruct binary patterns if they represent unusual features. If the image is partially broken, for example if some pixels



come from a broken sensor then  $\mathbf{N} \neq 0$ , or if some preprocessing generates some noise then  $\mathbf{Y}(\mathbf{P}) \neq 0$ , in both cases the binary descriptors will have insignificantly different values than the descriptors that have been used in the learning phase, they would be statistical outliers when compared to the training set of descriptors. In other words, in an RBM, they would generate high energy, hence low probability. Thus one can use the recurrent nature of an RBM to lower the energy of an input vector and obtain a binary descriptor closer to a training dataset. As explained, the generated  $\mathbf{B}$  matrix does not change, but in this case it will denote a signal with unknown noise, which the RBM will reduce. Consider (4.6), an RBM performs a single affine transformation followed by a non-linear activation function, so the RBM becomes de facto a feed-forward network. Taking advantage of the recurrent nature of an RBM it is possible to redefine  $\mathbf{\Gamma}(\cdot)$  to be a stochastic denoising function in the following way:

$$\begin{aligned} \mathbf{H} &\triangleq \sigma(\mathbf{W}\mathbf{B} + \mathbf{b}) > X \sim \mathcal{U}^{R_h \times R_v}, \\ \mathbf{E}(\mathbf{B}) = \bar{\mathbf{B}} &\triangleq \sigma(\mathbf{W}^\top \mathbf{H} + \mathbf{a}) > X \sim \mathcal{U}^{R_v \times R_h}, \\ \mathbf{\Gamma}(\mathbf{B}) &= \varphi(\mathbf{W}\bar{\mathbf{B}} + \mathbf{b}), \end{aligned} \tag{4.19}$$

$\bar{\mathbf{B}}$  in this case is a denoised matrix of binary descriptors and  $\mathbf{E}(\cdot)$  is a denoising function, also as can be observed the noised pattern initialises a Markov chain to have a denoised pattern as the output. This equation is simplified to one Gibbs step, however, similarly to Algorithm 1 there may be more steps of reconstruction which can be visualised as shown in Figure 4.9, where it is shown how an RBM may regenerate a noised vector to be a local attractor. Local attractor in this case means a configuration of an RBM's joints in a local energy minimum. The last reconstruction is then a reconstructed vector.

It is worth mentioning that this introduces an additional stage of processing which results in some additional latency, but despite this the overall preprocessing time should be still relatively short when compared to a CNN network which the preprocessing is followed by. The great advantage of this approach is that the denoising is not another module in the classification pipeline but just a feature that an RBM provides without any additional effort, so it can be enabled or disabled on demand after a training phase.

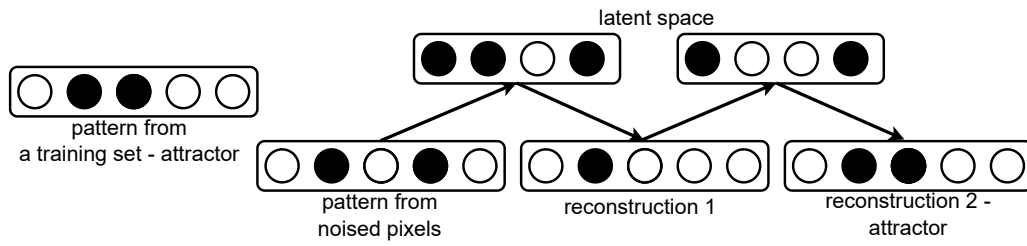


Figure 4.9: Pattern reconstruction in an RBM.

---

## EXPERIMENTAL RESEARCH

---

This chapter focuses on the empirical validation of the image processing methods that have been described in an earlier part of this dissertation. The major modification to a classical CNN pipeline proposed in this study is to use a preprocessing technique composed of a binary description layer and an RBM layer. Therefore, the experiments were mostly focused on testing how these layers affect some important indicators of classification quality. Experiments are split into four main sections. The first relates to image classification with an RBM's hidden units as feature space, the second to visual data comparison, the third to denoising the RBM's input, the last section relates to binary patterns aggregation and CD feature space. These are based on the three novel classification architectures proposed by the author in Chapter 4:

- **CD-KNN** - feature extractor based on contrastive divergence and descriptors aggregation (see Figure 4.1),
- **CD-CNN** - feature extractor based on contrastive divergence (see Figure 4.2),
- **HS-CNN** - feature extractor based on an RBM's hidden space (see Figure 4.6).

All the experiments were conducted with the use of the implementation of an RBM that was described in 4.1. This was used for training and for inference. For basic image processing, OpenCV library [163] was used. TensorFlow [153] was used as a tool for artificial intelligence purposes. Most of the code was implemented in Python, however, some parts of experiments were conducted with modules written in C++ and with CUDA [164] for GPU accelerated processing. Some basic matrix arithmetic operations were implemented with the use of NumPy [152]. Binary descriptors were implemented in OpenCV, except LBP since it has no implementation in this library and CLBP since it is a novel form of binary descriptor, thus these two had their

own implementation. The results were also partially presented by the author in [107, 165, 166].

The following datasets were used as images to train and test classifiers:

- Caltech101 [167]: dataset consisting of 9146 images split into 101 categories. Images are RGB and of different size. Numbers of images per category are unbalanced (40 to 800) and present different natural objects, like animals, faces, vehicles, devices etc.
- MNIST [168]: dataset consisting of 60000 images labelled and split into 10 categories. Images are Greyscale of size 28x28 and present handwritten digits.
- CIFAR10 [169]: dataset consisting of 60000 images labelled and split into 10 categories. Images are RGB of size 32x32 and present different animals and vehicles.
- STL-10 [170]: dataset prepared for unsupervised learning purposes. It consists of 10 categories each containing 500 images for training and 800 for validation. It also contains 100000 unlabelled images that can be used for unsupervised pretraining. Dataset is inspired by CIFAR-10, and samples are taken from ImageNet set. Images are RGB of size 96x96 and present similar objects to CIFAR-10.
- Indoor Scene [171] Dataset containing images presenting scenes from different indoor rooms. Images are RGB of different size.
- DTD [172]: dataset containing images presenting different textures. Samples are split into 47 categories, where each category contains 120 RGB images of different size.
- Concrete Cracks [173]: dataset containing images presenting concrete, split into 2 categories, the first contains samples with cracked concrete, the second samples with not-cracked concrete.

All the categories differ in some way. For example, MNIST is a dataset commonly considered as entry-level in image recognition, because the task to recognise the digits is relatively simple and even simple classifiers can achieve high performance on it and complex classifiers achieve accuracy close to 100%. However, it is a good example to test and improve new classification methods. CIFAR-10 is a well-prepared dataset because of the high number of samples per category but the size of the images is low, so learning features that can separate the classes may be challenging. Caltech101 presents objects

in a relatively clear context, they are aligned and not occluded which makes them easy to classify by neural classifiers, however, the number of images per category is not balanced and the number of classes is relatively high. STL-10, because of its categories composition it is a good case for testing an unsupervised learning method as this dissertation focuses on. Indoor Scene was chosen because of the nature of the objects it presents, it contains real world images, but they differ from other datasets in terms of what is visible in the scene. DTD was chosen because it contains artificial textures while others show real objects, so DTD is a dataset that differs the most. Concrete Cracks is a dataset containing more real case in terms of robotics, for example inspection mobile robots may be trained to recognise such defects in concrete. Some examples of images from all those datasets are shown in Table 5.1.

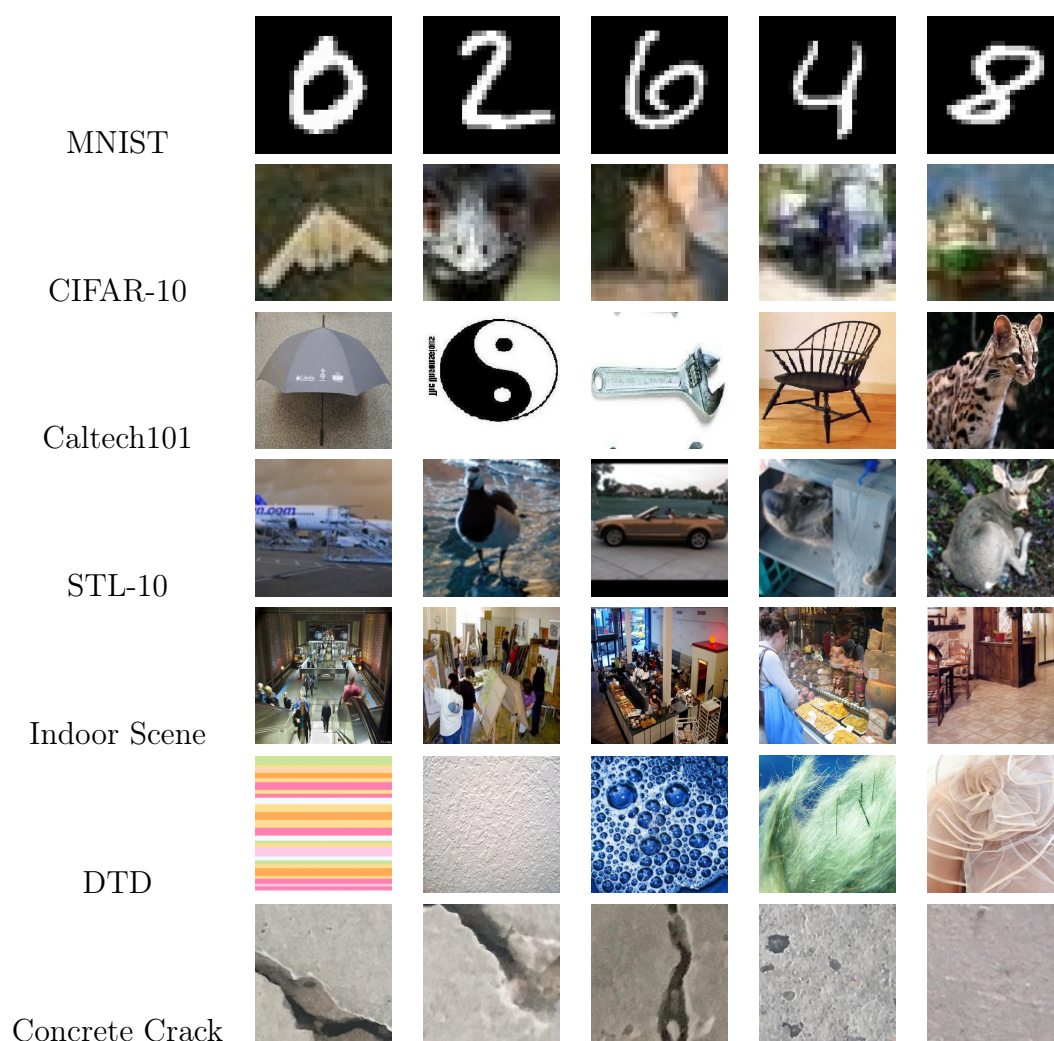


Table 5.1: Example images from different datasets used for experiments.

**Experiment 1 (CLBP / RBM implementation speed).** *This experiment was designed to test the speed of the author's implementation of CLBP and RBM.*

Before starting the analysis of binary descriptors and RBM capabilities in the context of image recognition it is worth investigating how fast their implementations work. Section 4.1 described details of how an RBM and a CLBP could be coded in python with the use of a GPU. It is clear that the time of response may depend on the size of an input image and other parameters. In the case of an RBM those parameters are the number of visible units and hidden units, for a CLBP one may investigate how fast it works when compared to a standard LBP8. The results of experiments with the speed are shown in Table 5.2 for a CLBP and in Figure 5.1 and 5.2 for an RBM. The conclusions are clear, the CLBP was not significantly slower than the LBP8, except for the largest tested size. The processing times were low, for all the resolutions they were not higher than 1ms, which made this layer very fast and applicable in real-time systems. The implementation of an RBM used in this research is also fast enough for lower resolutions and we could expect that the response time allowed applying it in real-time image processing systems, however for high resolutions, the times were relatively high. Thus the size of an input image may be chosen as a trade-off between speed and quality but it will be highly dependent on the particular task. The speed may depend on the internal implementation of matrix multiplications, as mentioned in section 4.1 the code written for the purposes of this research uses TensorFlow, it was designed for smaller resolutions as the used datasets are composed of images of lower resolution, for larger inputs it would be worth investigating other libraries. The second conclusion from analysing Figure 5.2 is that the response time of an RBM does not depend significantly on the number of hidden units, so this parameter may be adjusted to achieve the best classification accuracy. The number of visible units affects the speed more, this parameter is determined directly by the binary descriptors layer, and should be adjusted to achieve the best accuracy to response time ratio.

**Result of experiment 1.** *The implementation of CLBP and RBM prepared by the author provides short response times of these modules potentially allowing their use in systems with real-time capabilities.*

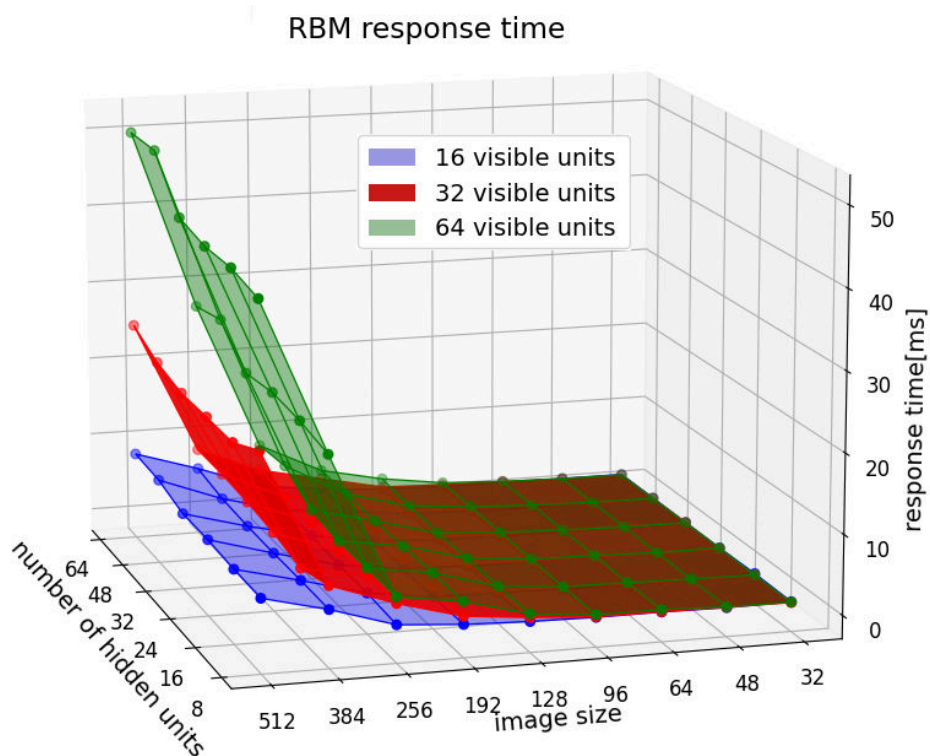


Figure 5.1: RBM response time depending on its parameter and size of an input image with a squared shape.

	input image size								
	32	48	64	96	128	196	256	384	512
LBP8	0.4	0.4	0.4	0.5	0.45	0.45	0.5	0.6	0.6
CLBP	0.4	0.4	0.4	0.5	0.45	0.45	0.5	0.7	0.95

Table 5.2: Time in milliseconds of LBP8/CLBP response depending on the size of an input image with a squared shape.



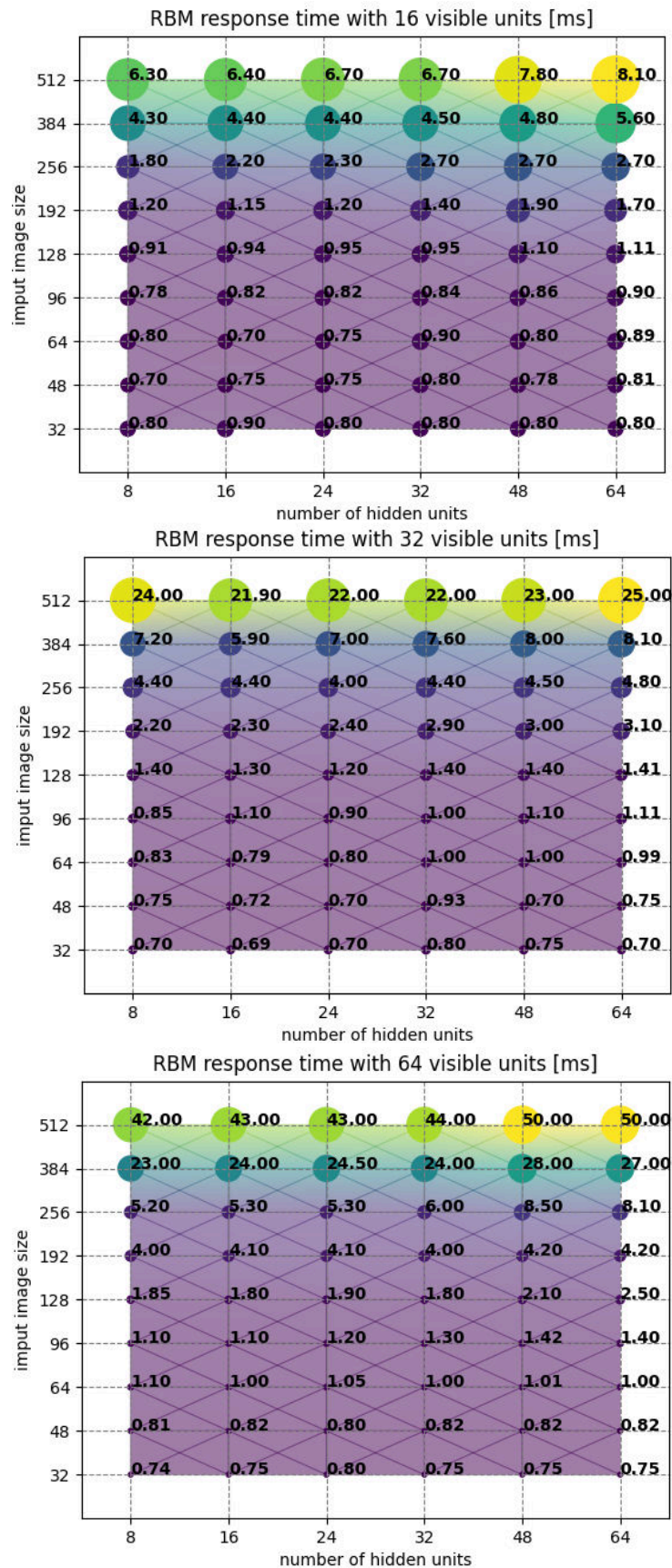


Figure 5.2: RBM Response time in milliseconds depending on the size of an input image with a squared shape and number of hidden units.

## 5.1 IMAGE CLASSIFICATION WITH RBM'S HIDDEN UNITS AS A FEATURE SPACE

---

### 5.1.1 LBP8-RBM - MNIST dataset experiments

---

**Experiment 2 (HS-CNN architecture basic capabilities).** *This experiment was designed to test how well this architecture performed when compared to widely known classifier on MNIST dataset.*

MNIST is a dataset often used to experimentally confirm the effectiveness of novel classification approaches. Sometimes, it is even referred to as a "Hello World" project in terms of image processing, which means it is a set of images that should be used to test if a given processing architecture works at all. Therefore it was used for validation of the efficiency of a LBP-RBM as a feature detector layer. The idea behind this experiment was to validate how a linear classifier could distinguish data from this extractor when compared to raw data. As the data in MNIST is composed of greyscale images of size 28x28 it is easy to train a neuron of input size equal to 784, which in experiments achieved a 92% accuracy and is a reference for this experiment. The results achieved by an LBP-RBM feature extractor followed by a similar classifier are shown in Figure 5.3. The experiment also included different parameters of an RBM: the number of stacked RBMs and an activation function. The networks were trained until the convergence, usually around 100 epochs, with the kernel size for the RBM set to 4. As can be observed, the network that achieved the best accuracy was composed of one RBM layer and used a hiperbolic tangent as an activation function. However, the most significant conclusion is that for the most of the RBM configurations, networks with preprocessing were trained with a higher accuracy than those without. Therefore it was experimentally confirmed that the proposed preprocessing can be applied as an image feature extractor.

Other parameters in an RBM that were adjusted: the number of hidden units, and the  $\beta$  scaling factor for the hiperbolic tangent. To infer the best ones, a dense grid searching could be performed, and the results of this are shown in Figure 5.4. The best set of parameters in this experiment were: the number of hidden units equal to 24, and  $\beta$  equal to 0.2. A number of other trends can be observed, setting  $\beta$  equal to 0.2 results in the highest accuracy for all numbers of hidden units, and increasing the number of hidden units resulted in a higher accuracy as long as it did not exceed 24.

It would be also possible to perform a more complex hyper-parameter tuning that took more possibilities of parameter configuration into account such as random search or Bayesian optimisation [174], and this could result in better

generalisation ability, however in the case of MNIST the purpose was to make the dependencies between some of the RBM's parameters clear.

Finally, for a tuned set of parameters the impact of RBM preprocessing was compared using different numbers of convolutional layers. The accuracy achieved by all the networks are shown in Table 5.3, and the learning curves are shown in Figure 5.5. Networks with RBM preprocessing achieved a better final accuracy, and usually converged faster. However with 3 convolutional layers, adding the preprocessing resulted in overfitting (validation loss started increasing while training loss remained decreasing), but this should not be considered as a problem since it can be solved by an early stopping or just by reducing the number of convolutional layers to 2.

**Result of experiment 2.** *HS-CNN architecture could be successfully applied for MNIST dataset classification and provided better generalisation ability than regular architectures for small number of convolutional layers.*

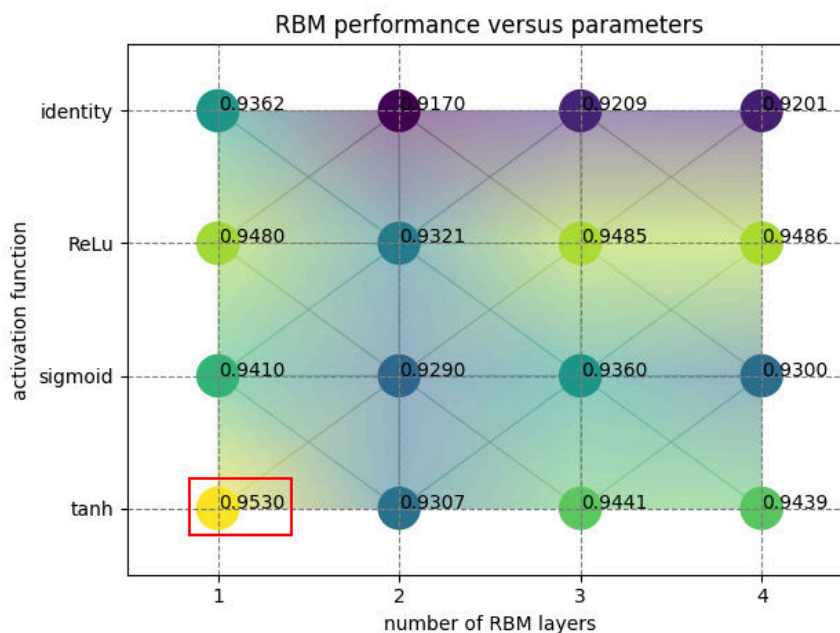


Figure 5.3: Results of experiments on MNIST dataset, a reference network without LBP-RBM achieves accuracy  $\approx 0.91 - 0.92$ .

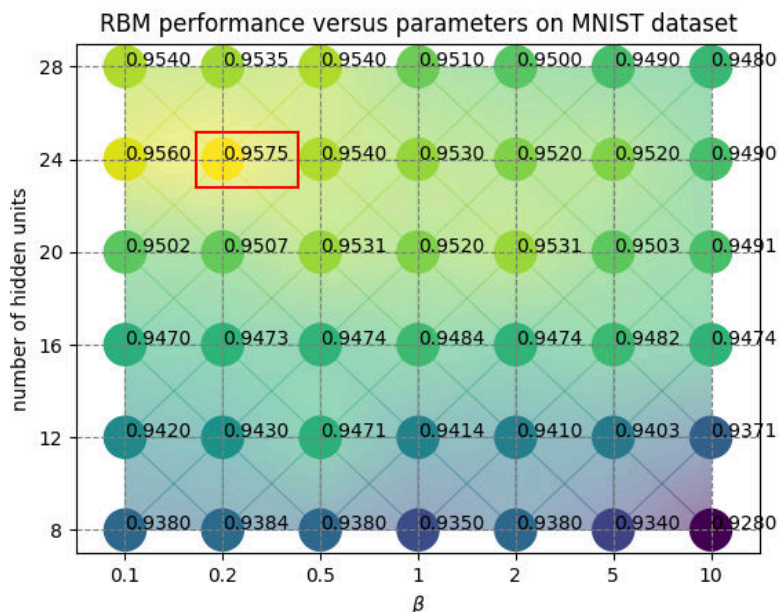


Figure 5.4: RBM hyper-parameters tuning on MNIST dataset.

	no. convolutional layers			
	0	1	2	3
RBM preprocessing	0.9575	0.9843	0.9916	0.9934
RAW pixels	0.9206	0.9732	0.9885	0.9915

Table 5.3: Accuracy achieved by neural network with different number of convolutional layers with and without RBM preprocessing on MNIST dataset.

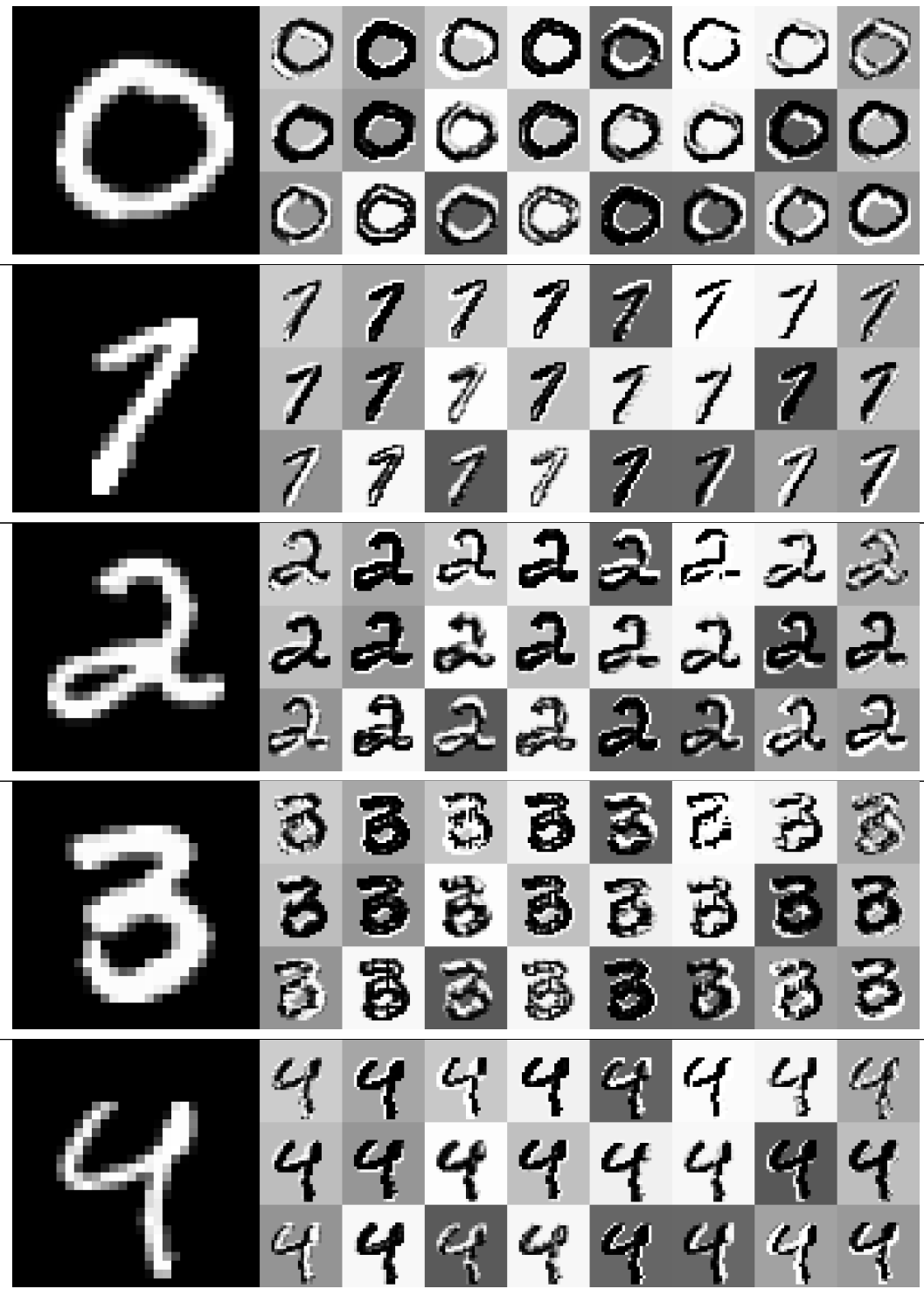
**Experiment 3 (Visualisation of learnt RBM's hidden space).** *This experiment was designed to visualise how RBM transformed images to its hidden feature space.*

For a trained RBM it is possible to visualise how it processes the image. In many cases with deep neural classifiers we can observe how the filters are learnt, meaning we know which features particular filters react to [19]. In the case of an RBM this is not useful since it does not process the raw pixel data, moreover the binary vectors that are not easily recognisable by a human. Instead, we can visualise the outputs from an RBM's hidden units for a given input image. Some examples are shown in Table 5.4. These outputs were generated with an RBM trained with a set of parameters that were previously determined as the best. It does not have to be obvious what

a particular output means, however, we can observe some clear relationships between an input and latent space features. For example the third hidden unit (first row, third column) reacted on the bottom and left edges, while the fifth hidden unit (first row, fifth column) reacted on the top and right edges, the sixth unit (first row, sixth column) enabled on the falling diagonal lines while the 11th unit behaved similarly but for rising diagonal lines, the 18th unit (third row, second column) was sensitive to everything which was not an edge. Thus, when the data was passed to a final classifier it had a more abstract meaning than raw pixels that denote only the intensity of a region in an image.

Finally, we can conclude that these experiments showed that LBP-RBM preprocessing can be successfully applied to an image classification problem.

**Result of experiment 3.** *RBM's hidden space could be visualised and interpreted in a relatively straightforward manner, and the dependencies between image features and RBM's unit's response could be observed.*



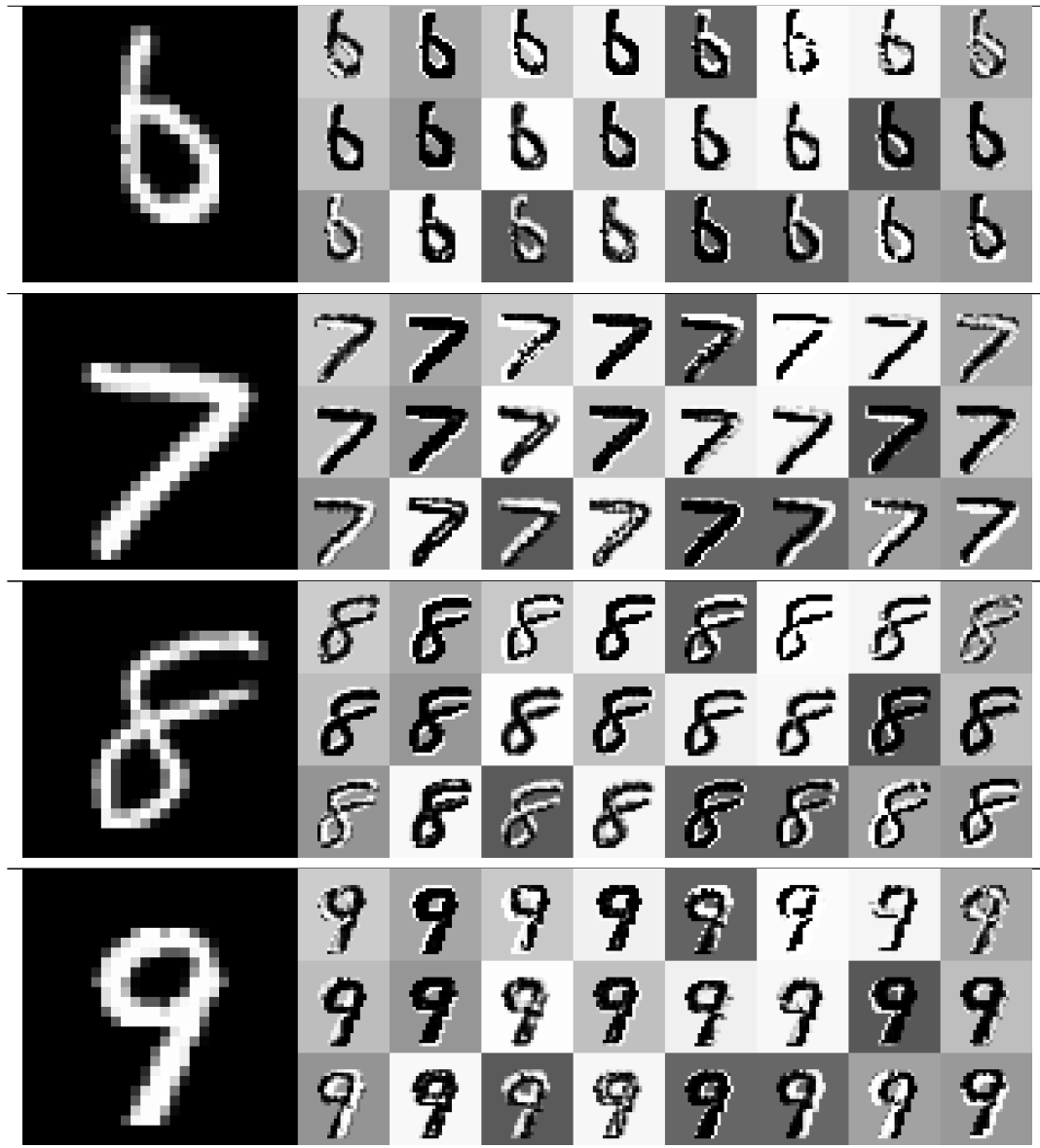
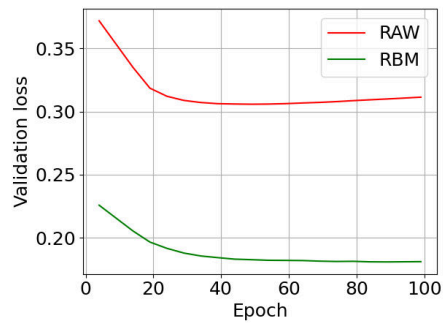
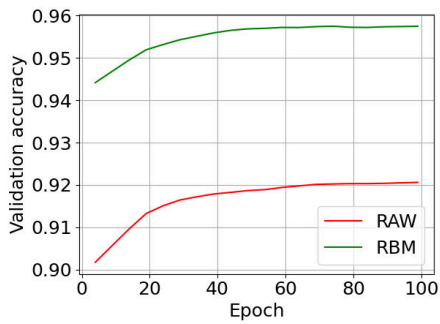
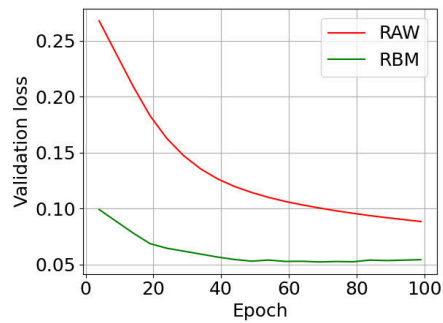
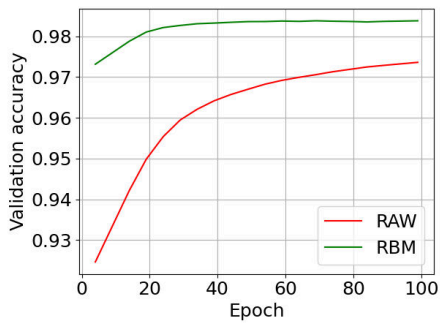


Table 5.4: Example outputs of RBM feature extractor. The left image in each row is an input image, the images on the right are outputs from each of 24 hidden units of an RBM.



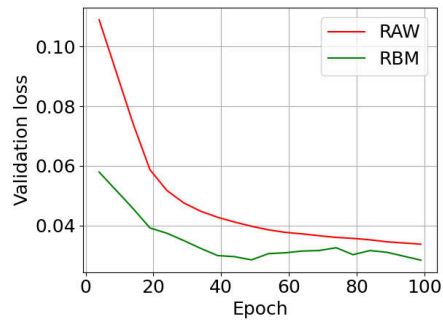
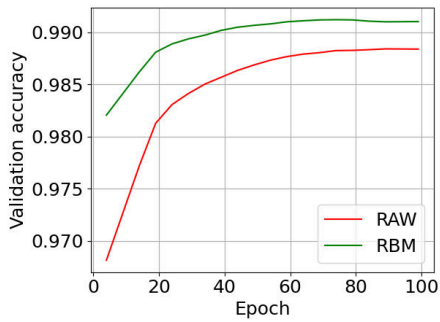
(a) Accuracy - 0 convolutional layers

(b) Loss - 0 convolutional layers



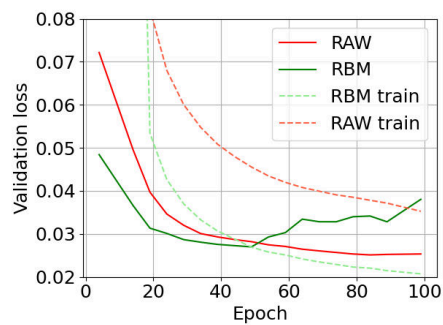
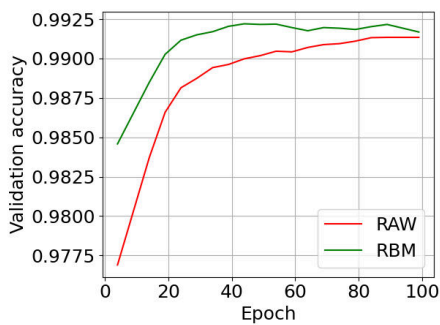
(c) Accuracy - 1 convolutional layer

(d) Loss - 1 convolutional layer



(e) Accuracy - 2 convolutional layers

(f) Loss - 2 convolutional layers



(g) Accuracy - 3 convolutional layers

(h) Loss - 3 convolutional layers

Figure 5.5: Validation learning curves with and without RBM preprocessing layer on MNIST dataset. Figure 5.5h contains training curves to emphasize the overfitting.



### 5.1.2 Colour Local Binary Pattern

---

**Experiment 4 (CLBP usability validation).** *This experiment was designed to validate how CLBP descriptor affected the classification ability in the **HS-CNN** architecture.*

The previous experiments were designed to use a regular LBP8 descriptor since the MNIST dataset is composed of greyscale images only. To validate how the proposed preprocessing works with colourful images the CIFAR-10 dataset was used. The dataset is composed of relatively simple categories, however, their size is low, which makes the classification task non-trivial. The first experiment was focused on verifying how adding the colour descriptor bits affected the network generalisation ability, thus the results were just a comparisons of curves for LBP8 and CLBP descriptors. Additionally, another descriptor referred to as "CLBP\_2" was added, which was formed similarly to a CLBP, but it did not include the additional 2 bits that denote the simplified intensity of a centre pixel. The results of this experiment are shown in Figure 5.6. The trends are clear, the CLBP achieved significantly higher accuracy than the LBP8 (4 percentage points on validation set after the optimisation convergence), and the CLBP also performed better than CLBP\_2. This implies that the proposition of an enhancement of the LBP descriptor may lead to a better classification accuracy when used in the proposed pipeline.

**Result of experiment 4.** *CLBP descriptor could be used in **HS-CNN** architecture and provided better generalisation ability than a regular LBP.*

---

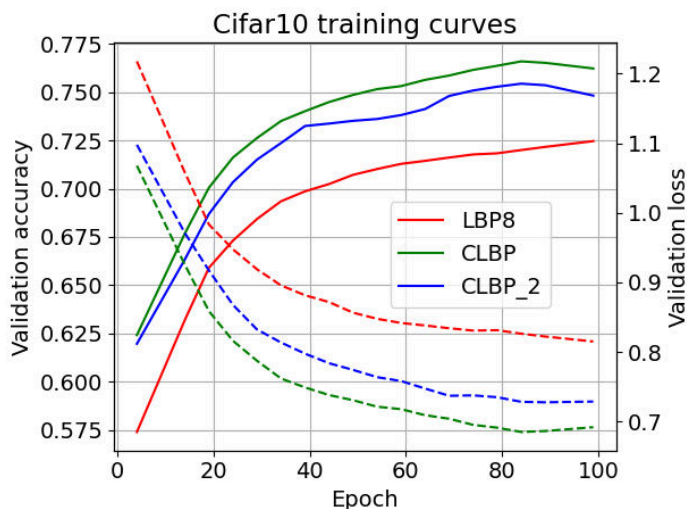


Figure 5.6: Accuracy on CIFAR-10 dataset depending on the type of descriptor used in preprocessing, "CLBP\_2" refers to CLBP without 2 intensity bits.

### 5.1.3 CLBP-RBM - CIFAR-10, STL-10, Concrete Cracks experiments

The experiments described in the previous sections indicated the validity of the assumption that the CLBP-RBM preprocessing may be applied to image recognition and that the novel CLBP descriptor provides better classification quality than a regular LBP. The experiments described in this section aimed to confirm the effectiveness of the proposed method in more complex problems and to generalise the conclusions based on more datasets. A neural model composed of layers as in Figure 5.7 was created for the classification task. This also introduced two additional stages of processing to **HS-CNN** architecture, first was a Global Average Pooling (GAP) [182] which optimised the learning processes to generate feature maps that represented a particular input category, the second was a Dropout [149] that prevented overfitting.

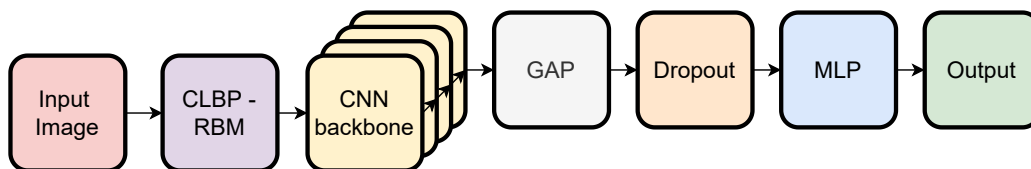


Figure 5.7: Image classification pipeline for CIFAR-10, STL-10 and Concrete Cracks datasets, **HS-CNN** architecture enhanced with Dropout and GAP.

**Experiment 5 (RBM hyper-parameters optimisation for CLBP descriptor).** *This experiment aimed to adjust the number of RBM hidden units and kernel size for the best performance based on the STL-10 dataset.*

To begin with, the important part of the experiment was to set hyper-parameters of an RBM, in fact, it was done earlier for MNIST, however the optimality of some parameters may differ in case of preprocessing another descriptors (LBP8 for MNIST, CLBP for STL-10). The number of RBM layers and the type of activation function were valid because the extraction principles were the same, but the best number of hidden units could differ as the descriptor sizes differed. Also as the input images differed in size the best kernel size also had to be obtained, this was set to 2 for MNIST since the images were very small. These experiments were intended to represent a real use case, so processing time also had to be taken into account. Therefore to adjust the number of hidden units and the kernel size, as a criterion of optimality the following formula was used:

$$o \triangleq \frac{\text{accuracy}}{\text{processing time}}, \quad (5.1)$$

where accuracy and processing time values were relative to the lowest achieved accuracy and the highest measured processing time. The experiments were conducted on the STL-10 dataset. RBMs were trained over 30 epochs in each experiment on unlabelled data, then the backbone was trained with an RMSProp optimisation algorithm until reaching convergence. The results are shown in Figure 5.8. One can observe that the best  $o$  was achievable with the use of kernel size equal to 2, and the number of hidden units to 48, thus those values were used for the rest of the experiments.

**Result of experiment 5.** *RBM achieved the best performance for kernel size equal to 2 and the number of hidden units equal to 48.*

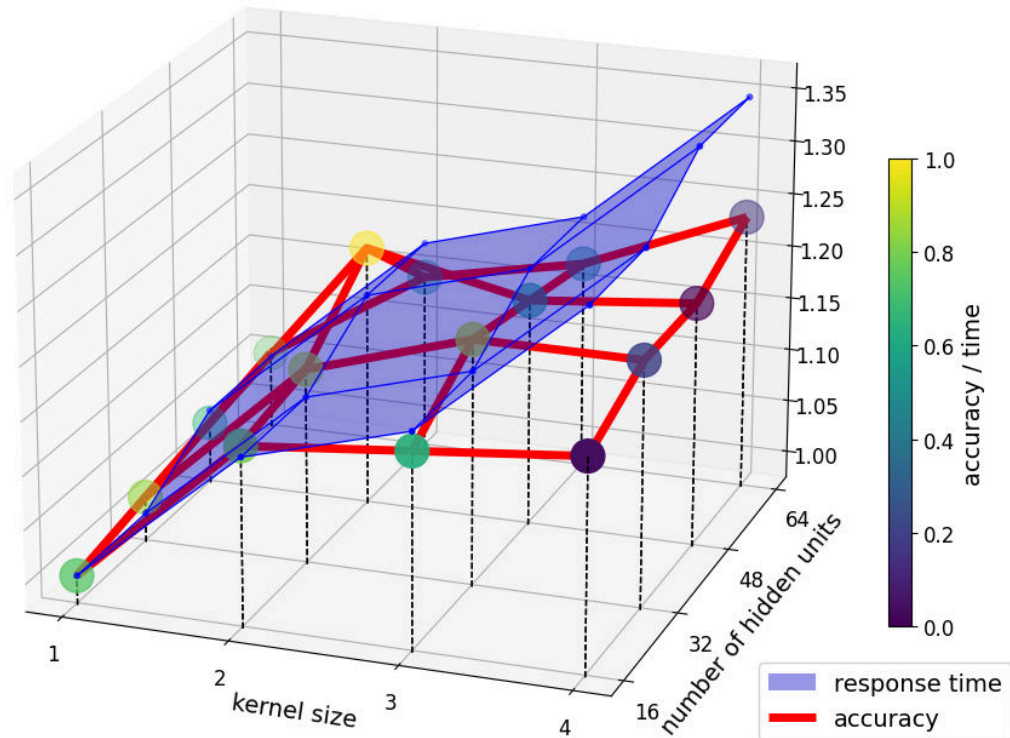


Figure 5.8: RBM performance versus its hyper-parameters. Results are relative to the lowest accuracy and to the highest processing time.

### Experiment 6 (RBM latent space visualisation for CLBP input).

*This experiment aimed to visualise RBM's hidden space when processing CLPB input based on trained on the STL-10.*

Once the parameters were adjusted, the main experiment that was designed to investigate the impact of adding the CLBP-RBM layer was conducted. For a visualisation of how an RBM processes colour images a similar experiment to that of the MNIST experiment was conducted, so an example image was processed by an RBM and the features from each hidden unit are shown as an image, however in this case it is worth analysing what the same features look like before an RBM was trained. The features in an untrained RBM are shown in Figure 5.9 and in a trained RBM in Figure 5.10. Both images illustrate the hidden features of each channel. The features generated by an untrained RBM were without any relation to the input image, thus were not useful in terms of further processing, which was in contrast to the trained RBM response where we can observe how some hidden features were related to the particular input features. Some of the visible connections are highlighted. For example, the blue rectangles mark vertical lines in the in-

put image and the filter that reacted on these. The filter marked with green colour reacted on vertical and horizontal lines. Red rectangles mark regions containing red colour and the filter reactive to that. Similarly, yellow regions mark filters reactions to green colour.

**Result of experiment 6.** *RBM hidden space sensitivity to colour and shape of features was observable.*

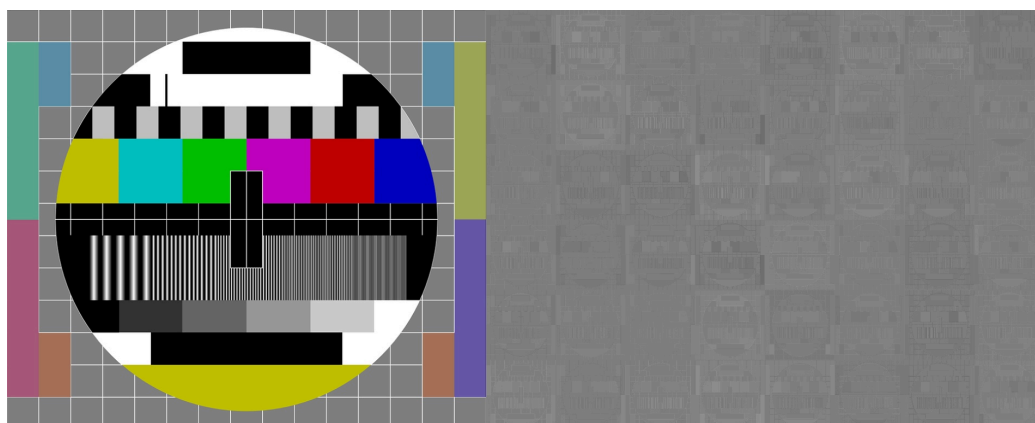


Figure 5.9: RBM's hidden space features representation in an untrained model.

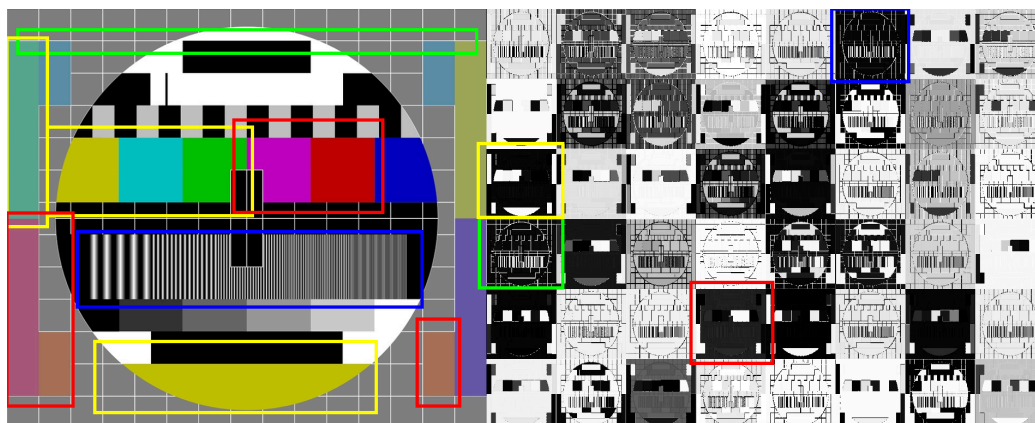


Figure 5.10: RBM's hidden space features representation in a trained model.

**Experiment 7 (HS-CNN generalisation ability validation).** *This experiment aimed at testing HS-CNN architecture classification performance on the STL-10 dataset. This was the most important experiment in terms of large scale image classification in this study.*

After these initial experiments, the final tests that related to the exact classification performance in the proposed processing pipeline were conducted. Three commonly known deep learning architectures were chosen as CNN backbones: VGG, DenseNet and ResNet, they differ in terms of complexity and internal connections, thus they were good examples to make general conclusions. Also, as these CNNs are relatively complex networks, another small network was added, it was composed of six convolutional layers followed by max pooling, which will be referred to as "custom". The training details related to CNN are described in Appendix B. The results achieved by those networks are shown in Table 5.5, where we can observe how the accuracy changed when the CLBP-RBM was used and when it was not. The table also contains the number of parameters in those networks and their processing time. Based on those numbers one may conclude that the preprocessing layer improved the accuracy in any case independently of the size of the network, and also for each type of backbone the processing time of the CLBP-RBM was relatively short when compared to the rest of the network. This implies that adding a small preprocessing network may result in increasing the final network accuracy without significantly decreasing the network speed.

This is one of the most important results in this dissertation, since the research is based primarily on the theorem that unsupervised trained RBMs introduce an additional level of abstraction, without being very complex. Based on that assumption it was possible to perform another experiment, one may claim that the CLBP-RBM layer may replace some number of conventional layers, subsequently, a deep neural network with preprocessing should achieve greater accuracy when the number of convolutional layers is reduced when compared to a network without it. Thus the experiment was based on validation how reducing number of convolutional layers affects the accuracy of classification model. Table 5.6 contains the results, the impact of the preprocessing is clearly observable, a model containing the CLBP-RBM layer may be shrunk with a smaller decrease in performance than a model without this layer. Figure 5.11 illustrates the learning curves gathered during the CNN training phases, one may notice the overfitting, which was probably caused by the relatively low number of training samples for a relatively complex network, however this effect was less noticeably visible in models with preprocessing, and also the "custom" network had not been significantly

overfitted. Therefore for the experiments in the next sections this type of backbone would be used.

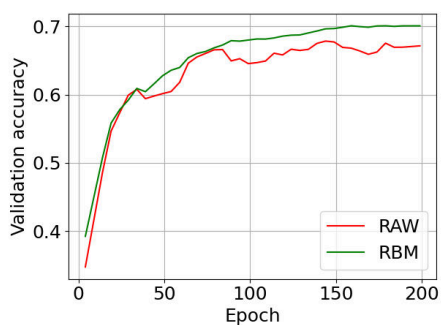
**Result of experiment 7.** *HS-CNN pipeline achieved greater accuracy for various CNN backbones than a network without the CLBP-RBM preprocessing.*

	backbone type			
	custom	VGG	ResNet	DenseNet
Accuracy improvement	2.6%	2.9%	4.7%	7.5%
Number of parameters	2.76M	16M	27.7M	7M
Backbone processing time - CLBP-RBM relative	15x	22x	135x	300x

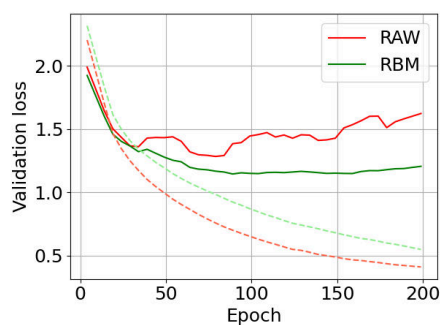
Table 5.5: Accuracy improvements achieved by different types of network backbones with the CLBP-RBM preprocessing.

	no. convolutional layers reduction			
	1	2	3	4
CLBP-RBM	<b>2.50%</b>	<b>5.00%</b>	<b>19.00%</b>	<b>25.00%</b>
no preprocessing	4.10%	10.00%	25.00%	37.00%

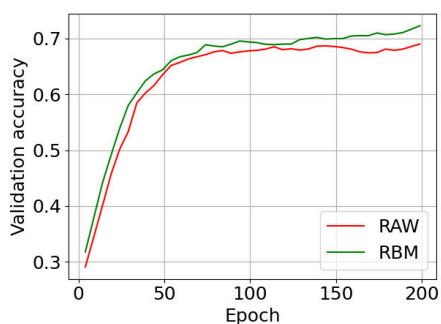
Table 5.6: Accuracy reduction versus number of CNN layers reduction in "custom" backbone.



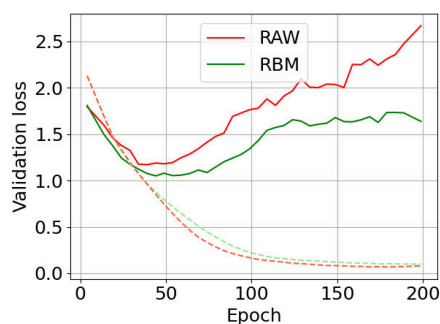
(a) Accuracy - "custom" backbone



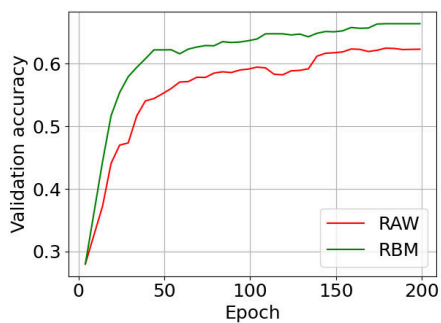
(b) Loss - "custom" backbone



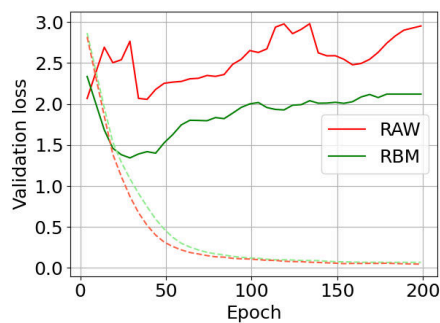
(c) Accuracy - VGG backbone



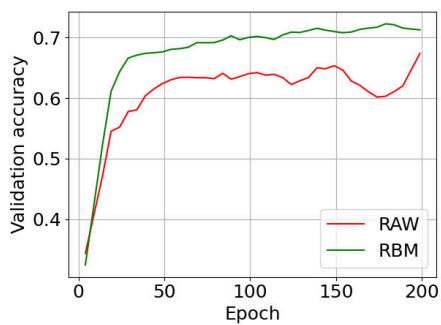
(d) Loss - VGG backbone



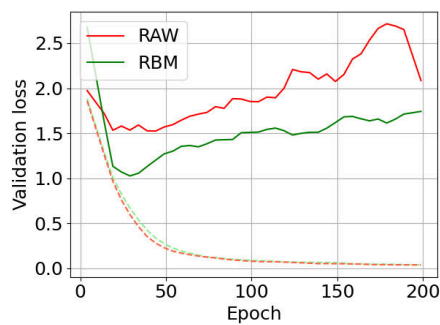
(e) Accuracy - ResNet backbone



(f) Loss - ResNet backbone



(g) Accuracy - DenseNet backbone



(h) Loss - DenseNet backbone

Figure 5.11: Learning curves with and without the CLBP-RBM preprocessing layer on the STL-10 dataset. Dashed lines denote results on training subset, solid lines on validation subset.



**Experiment 8 (HS-CNN generalisation depending on the number of training samples).** *This experiment was designed to validate how the size of a training dataset affected the **HS-CNN** classification accuracy when compared to a network without the **CLBP-RBM**.*

STL-10 is a dataset composed for testing unsupervised learning methods because of the large disproportionality between the training dataset and the testing dataset, whereas CIFAR-10 is a dataset that can be successfully used for CNNs that are trained with supervision. Thus the last experiment in this section validated how the number of training samples affected the final accuracy in the case of training a CNN with and without the CLBP-RBM preprocessing. The idea of the experiment was simple, the initial number of training images was 50000, and the number of testing images was 10000. The "custom" network was trained on this set to verify the accuracy, and then the size of the training set was decreased to be respectively 75%, 50%, 25%, 10%, 5%, 1% of the initial size, the tested metrics was verified and compared for the networks with and without preprocessing. The results are shown in Figure 5.12. The first important thing to note is that when all training samples from CIFAR-10 were used for training a CNN, the preprocessing did not affect the final accuracy and both types of networks achieved similar results. This was because this dataset is well composed, the number of images per each category is well balanced and diversified, so even a simple deep neural network did not need any preprocessing to learn to categorise these images. However, if the size of the training dataset was decreased the impact of the CLBP-RBM preprocessing became more significant. One can observe that the accuracy decreased when the number of training images decreased, but this was not surprising. It was claimed at the beginning of this dissertation that deep neural networks need a large number of training samples in order to tune properly their parameters. However, this experiment was designed to show the importance of the CLBP-RBM layer and it is clearly observable in the results. The decrease in accuracy was lower for each of the tested sizes of the training set, and the difference between the generalisation ability was higher as the size decreased. Therefore one can conclude that adding the CLBP-RBM layer may result in increasing the accuracy ability of a neural network when the availability of training data is limited, and it does not affect this metrics when the dataset is balanced and well prepared.

**Result of experiment 8.** *HS-CNN needed fewer training samples than a network without the CLBP-RBM to achieve the same accuracy.*

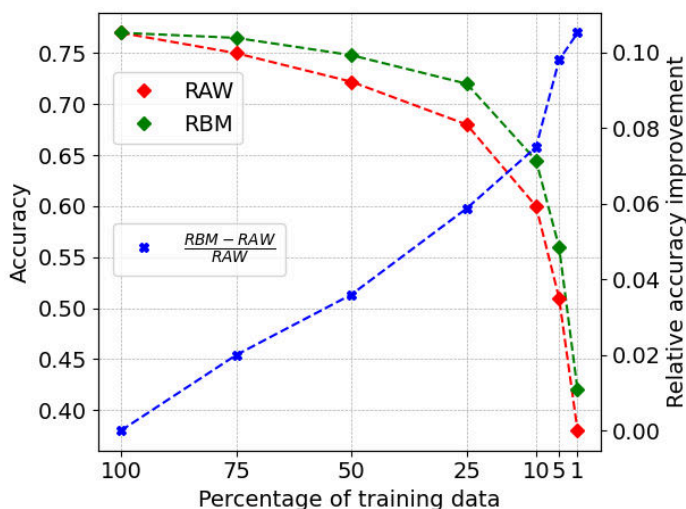


Figure 5.12: Evaluation of deep neural network performance depending on the number of training samples with and without the CLBP-RBM preprocessing on CIFAR-10 dataset.

**Experiment 9 (HS-CNN for concrete crack detection).** *This experiment was designed to validate how the proposed preprocessing method affected generalisation ability in concrete cracks detection task.*

The previous experiments based on some widely used datasets showed that the proposed preprocessing method improves classification accuracy in some scenarios. Therefore, this test aimed to verify how this can be moved to a more realistic case. A good example is the Concrete Crack detection, which may be performed by a mobile robot and other climbing machines. It turns out that this type of classification task is a common challenge in robotics and various applications of robots to support remote/autonomous inspection of building structures, road surfaces, etc. are reported in the literature [183, 184, 185, 186, 187]. The previously mentioned dataset [173] contains images of concrete split into two categories: with ("crack") and without cracks ("concrete"). However, to simulate more realistic scenario, another class was added. The additional category ("others") was built from images from the Caltech101, DTD, Indoor Scene datasets and was composed of images that did not present any concrete. The dataset was balanced and contained 38k samples for training, 10k samples for validation and 12k for testing. The three-category classifier can be applied in real device to detect if camera is processing frames from concrete, cracked concrete or a different scene. The results of evaluation on the testing for both networks are shown

in Table 5.7, the accuracy versus percentage of training data is shown in Figure 5.13. We can notice, that for each of the testing metrics there was an improvement when the CLBP-RBM layer was applied, which implies that this type of classification architecture may be employed to solve real image recognition problems, which for example can be encountered in some robotic applications.

	Crack		No crack		Others	
	RBM	RAW	RBM	RAW	RBM	RAW
Accuracy ( $A$ )	0.996	0.986	0.991	0.989	0.992	0.978
Precision ( $P$ )	0.997	0.998	0.997	0.995	0.977	0.942
Recall ( $R$ )	0.991	0.960	0.978	0.971	0.999	0.996
Specificity ( $S$ )	0.999	0.999	0.998	0.999	0.987	0.967
$\frac{\sum_{\{A,P,R,S\}} (RBM-RAW)}{4}$	0.001		0.00275		0.0178	

Table 5.7: Evaluation of recognition pipeline generalisation ability metrics for each classified category on the testing dataset for concrete crack detection.

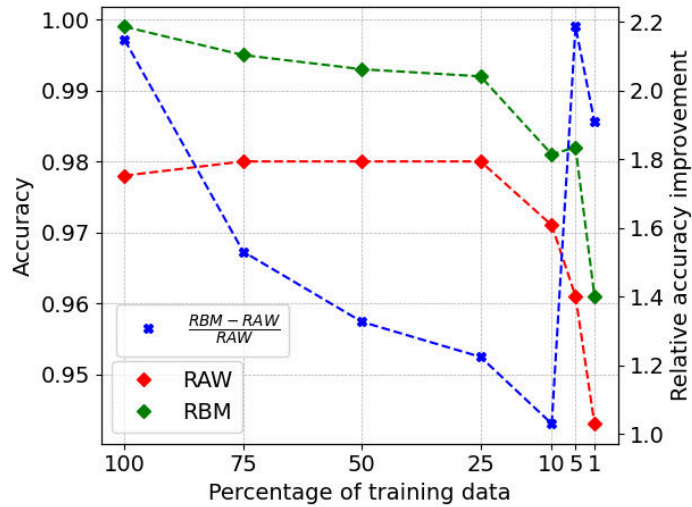


Figure 5.13: Evaluation of deep neural network performance on all categories depending on the number of training samples with and without the CLBP-RBM preprocessing on Concrete Crack dataset.

**Result of experiment 9.** *The HS-CNN architecture achieved higher efficiency metrics than the network without preprocessing on Concrete Crack dataset.*

## 5.2 CLBP-RBM FOR VISUAL DATA COMPARISON

---

### **Experiment 10 (Visual data comparison with the CLBP-RBM).**

*This experiment was designed to validate the proposed method for visual data comparison.*

In section 4.4 presented the novel idea of how an RBM can be used to compare visual data that can be an input to a conventional neural network. The similarity of the data can be very subjective in terms of what one can expect of the metrics that is the output of the comparison. For neural network purposes such metrics should inform how close a given images dataset is to another, particularly to one that has been used to train a preprocessing layer. This section describes an experiment that took advantage of an RBM for such a comparison. Three RBMs were trained on STL-10, Indoor Scene and DTD datasets, then for each one, two other datasets were used to validate the similarity metrics. For humans, the DTD differs from the other two in terms of the objects it presents, the images in it are artificial or are taken with a close shot from a textured objects. The other two present real objects, thus intuitively they should be similar for a neural network. Therefore one may expect that the proposed metrics should be lower for Indoor Scene and STL-10 comparison while higher for a comparison with the DTD. The results of this experiment are shown in Figure 5.14 and Table 5.8. For each of three RBMs histograms were generated with the use of a subset of each dataset. The reference histograms were flattened, and there were only 10 bins for the sake of better readability. A fourth histogram for each RBM generated from another subset of the training set was added, referred to with the suffix "2". One may observe that for each case the distances to these sets of images were the lowest, which was not surprising because they present the most similar visual features, however, it confirmed experimentally that the assumption of using this type of similarity measure was valid. The more important conclusions are made when analysing the distances to other datasets. For an RBM trained on STL-10 data the distance to Indoor Scene dataset was significantly lower than to DTD, analogically, RBM trained on Indoor Scene showed to be closer to STL-10 than to Indoor Scene. Finally an RBM trained on DTD revealed high distance to both other datasets. All these findings lead to the conclusion, that the proposed method for analysing the similarity of datasets is effective because the achieved results are close to the expectations in the context of what is similar for humans.

**Result of experiment 10.** *CLBP-RBM based histograms of probabilities can be used to compare the similarity of image datasets.*

	Data STL	Data Scene	Data DTD
RBM STL	0.0015	0.0193	0.0601
RBM Scene	0.0054	0.0001	0.0339
RBM DTD	0.0349	0.0264	0.0004

Table 5.8: Chi-Squared distances (defined by (4.18)) between datasets measured with the use of an RBM’s probabilities histograms, the diagonal results denote the distance between the reference set of images and another set from the same training dataset.

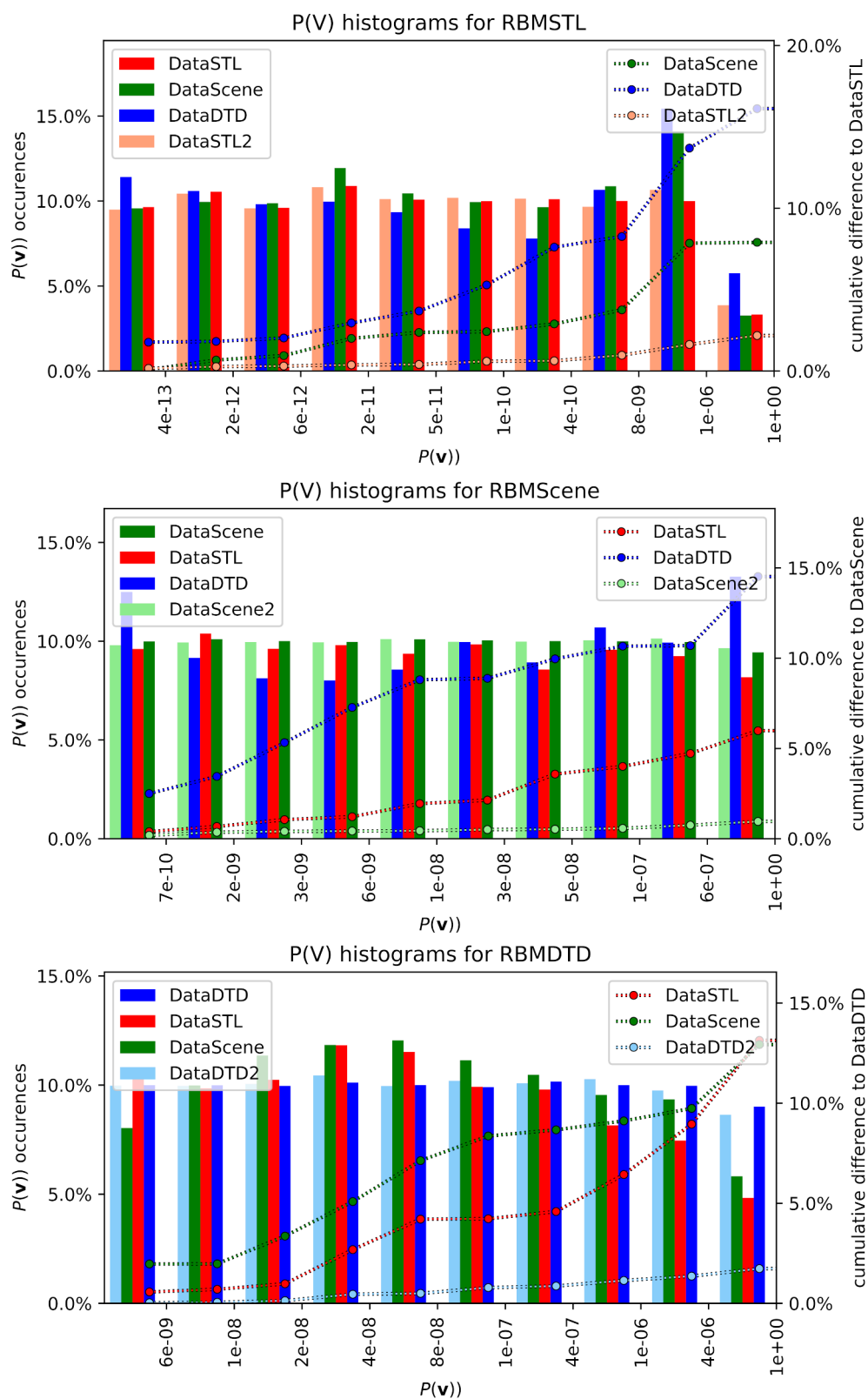


Figure 5.14: Histograms for  $P(\mathbf{v})$  taken from different datasets, cumulative difference denotes the sum of differences between a given bin to a reference bin from the first to the current bin.

---

## 5.3 CLBP-RBM FOR VISUAL INPUT DENOISING

---

### 5.3.1 Binary vectors denoising

---

**Experiment 11 (Binary descriptors denoising with RBMs).** *This experiment was designed to validate if RBMs can be used to reconstruct binary vectors being image descriptors.*

The idea of using the CLBP-RBM layer for denoising an input to a CNN was presented in Section 4.5 where it was explained how the recurrent nature of an RBM can be used to reconstruct a binary features vector. This section focuses on testing the proposed approach. Since a CLBP can be considered a non-linear non-invertible map, it is not possible to make a reconstruction and visualisation of a raw input image, however, it is possible to reconstruct a binary string. Therefore, the first step in investigating this denoising solution was choosing the best number of recurrence iterations in an RBM. To do that, an RBM was trained on the STL-10 dataset, since in that instance the time of response was not crucial, the kernel size was set to 4. Then, to have a noised dataset, 50 samples for a CLBP input were generated from the STL-10 dataset, and  $n$  number of their pixels were randomly noised with the use of the following formula:

$$\forall_{x \in \{1,2,3\}} p_{i,j,x} = 255 \text{ if } X_{\sim \mathcal{N}} > 0.5 \text{ else } p_{i,j,x},$$

where  $i, j$  are the coordinates of the randomly chosen pixel to noise and  $p_{i,j,:} \in \{0, 1, \dots, 255\}$ . Some examples are presented in Table 5.10, where we can observe how the distortion affected the images being an input to the CLBP. For a kernel size equal to 4 the number of bits in the CLBP was equal to 256. To measure the similarity of descriptors Hamming distance [122] was used, and a metrics to measure the reconstruction ability was defined by this formula:

$$o \triangleq \underbrace{D(\zeta(\mathbf{P}), \zeta(\tilde{\mathbf{P}}))}_{\text{original-noised distance}} - \underbrace{D(\zeta(\mathbf{P}), \epsilon(\zeta(\tilde{\mathbf{P}})))}_{\text{original-denoised distance}}, \quad (5.2)$$

where  $D(\cdot)$  denotes a Hamming distance function,  $\tilde{\mathbf{P}}$  is a noised image generated from  $\mathbf{P}$ ,  $\zeta(\cdot)$  - CLBP transformation,  $\epsilon(\cdot)$  RBM denoising function. Formally the  $o$  metrics informs as to how well an RBM reconstructed the original pattern, an ideal case is when the second term equals 0, the first term was bigger than zero in all the test cases. The comparisons were carried out 100 times on each of 50 samples, the  $o$  values were averaged, and the results are presented in Table 5.9. One may observe that the metrics increased as the noise factor increased due to characteristics of the first term

---

in (5.2). This then indicates the validity of applying the metric, but the values may be compared only within the same noise factor. In conclusion, the ideal number of Gibbs steps for reconstruction was 1 for all the tested scenarios as the  $o$  value was lowest for this number for each noise factor, hence it would be used for further experiments.

		noised pixels			
		1	2	3	4
Gibbs Steps	1	<b>1.24</b>	<b>7.46</b>	<b>13.6</b>	<b>19.2</b>
	2	-0.7	5.5	11.23	16.713
	3	-1.77	4.21	10.04	15.44
	4	-2.47	3.38	9.28	14.96

Table 5.9: Reconstruction results.

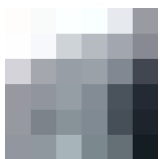
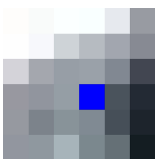

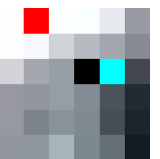
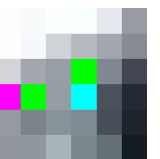


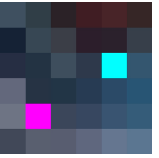
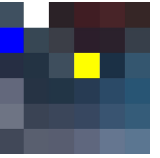
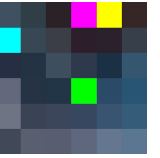
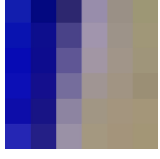
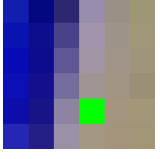
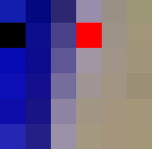
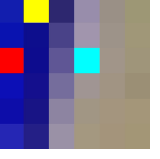

Nr	original	n=1	n=2	n=3	n=4
1					
2					
3					

Table 5.10: Example patterns used for reconstruction, the original and with added random noise.

After the parameters of an RBM were defined, an experiment that relied on applying the reconstruction phase for classification purposes was conducted. The idea was relatively simple, an RBM was trained on the STL-10 dataset in the same manner as in previous experiments. Then a CNN was trained on a training dataset without any noise, but the validation phase was performed with the use of distorted images. In the first experiment the distortion had a similar nature to as it was in the case of testing the best



number of Gibbs steps, so some number of pixels in the image were randomly noised. The noise factor was defined as a ratio between the number of noised pixels and the number of all pixels in the image. Examples of images with this type of distortion are presented in Figure 5.15. The experiments were conducted for 4 different noise factors (0, 0.025, 0.050, 0.075), as a result learning curves were presented for all these cases, as in Figure 5.16. We can analyse how an RBM learnt the data when denoising was enabled and when it was not, and also compare to results without the CLBP-RBM preprocessing. The first finding is that enabling the denoising did not negatively affect the classification accuracy even if there was no noise added. Then, when the noise factor was small (2.5% noised pixels) reconstructing the binary patterns did not improve the generalisation ability, however, removing the CLBP-RBM layer reduced it significantly. For bigger noise factors, the difference between the denoising enabled and disabled was clearly observable as well as the accuracy reduction when there was no preprocessing layer. In conclusion, running the reconstruction method in the CLBP-RBM may be advantageous when the classification images are expected to be noised and not disadvantageous when they are not. The only negative aspect is the recurrence causing an RBM stage to be slower, approximately 2.5 times, however it is probably not a major concern since the time of response was for most cases lower than 5 ms, which made this fast enough even when the denoising was enabled.

**Result of experiment 11.** *The CLBP-RBM can be used for binary vectors reconstruction.*

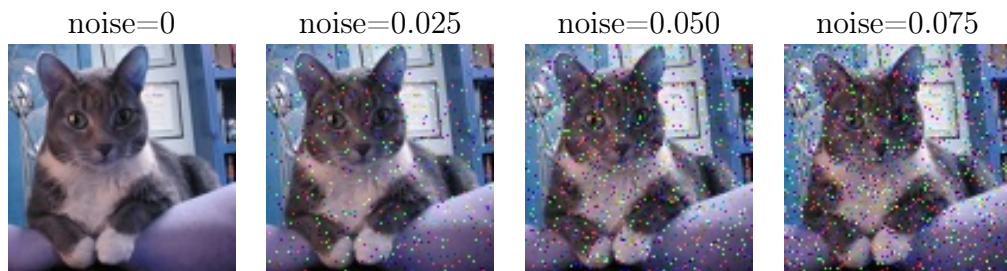
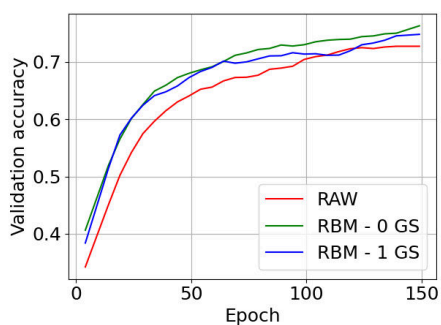
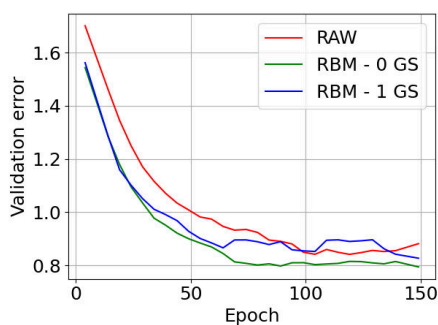


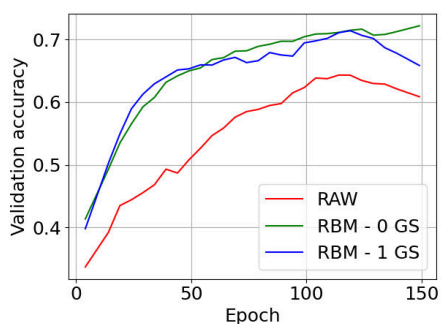
Figure 5.15: Examples of noised images with different noise factors.



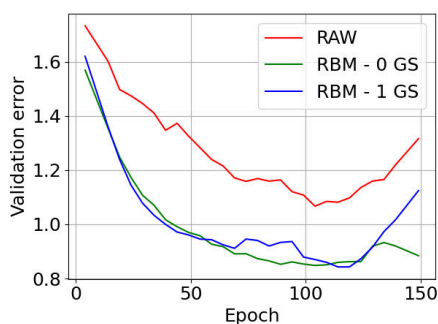
(a) Accuracy - noise factor = 0.



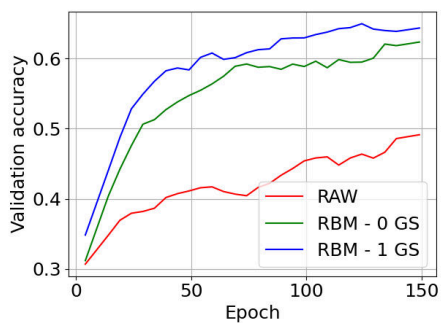
(b) Loss - noise factor = 0.



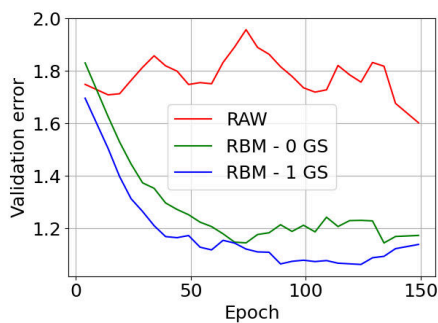
(c) Accuracy - noise factor = 0.025.



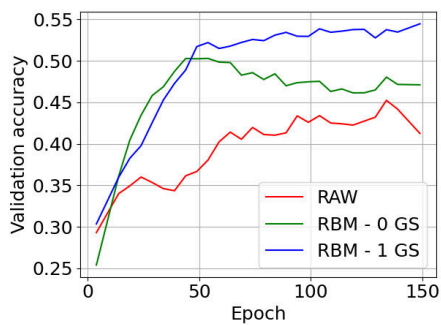
(d) Loss - noise factor = 0.025.



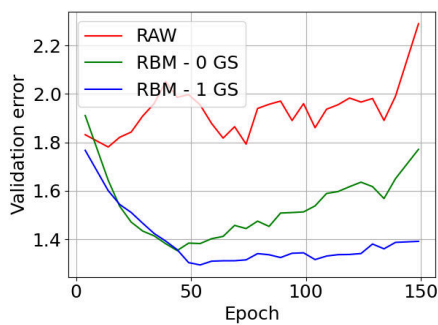
(e) Accuracy - noise factor = 0.050.



(f) Loss - noise factor = 0.050.



(g) Accuracy - noise factor = 0.075.



(h) Loss - noise factor = 0.075.

Figure 5.16: Validation learning curves for a denoising experiment, "GS" denotes number of Gibbs steps performed for reconstruction.

### 5.3.2 Adversarial attack

#### Experiment 12 (Adversarial attack sensitivity of the HS-CNN model).

*This experiment was designed to validate if reconstruction of binary vectors results in better generalisation ability for gradient based noise.*

Another type of noise that may be applied to an image and cause a lowering of the generalisation ability is the fast gradient sign method referred to also as adversarial attack [188]. This type of distortion is interesting due to it being unobservable by humans, which means that human is not able to recognise whether a picture was noised or not but a classification neural network may fail during processing such sample and predict an invalid category. The example of such a case is presented in Figure 5.17, where it can be observed how the distortion affects the prediction. Technically this method is defined by the following equation:

$$\tilde{\mathbf{P}} = \mathbf{P} + \epsilon \cdot (\nabla_{\mathbf{P}} J(\theta, \mathbf{P}, y)), \quad (5.3)$$

where  $\epsilon$  is a noise factor. The noise is then generated based on the gradient propagated over the model with the use of output from the original image. This makes this technique applicable only with a knowledge of the model, in contrast to the previous technique.



(a) Prediction - "dog" - 99%.

(b) Prediction - "cat" 92%.

Figure 5.17: Examples of an image with (5.17b) and without (5.17a) added adversarial attack noise.

Intuitively, the CLBP-RBM may be insusceptible to this type of image perturbation for the same reason as it was in the case of random noise. The

preprocessing layer can make a reconstruction of an input, so the distortion may be removed or at least lowered. The basic experiment to validate this hypothesis was comparison of how adding the CLBP-RBM layer affected the generalisation ability of a neural network when input images were noised with adversarial attack. It was performed for different values of  $\epsilon$ , and the results are shown in Figure 5.18. The noise affected both networks, however, the model with a preprocessing layer was significantly less prone to this type of perturbation. This shows that the CLBP-RBM layer is a good solution for deep neural networks that have to handle noised data, especially since the same type of denoising can be applied for different types of noise.

**Result of experiment 12.** *Adding the CLBP-RBM preprocessing to a CNN network resulted in better generalisation ability when input images were perturbed with a gradient based noise.*

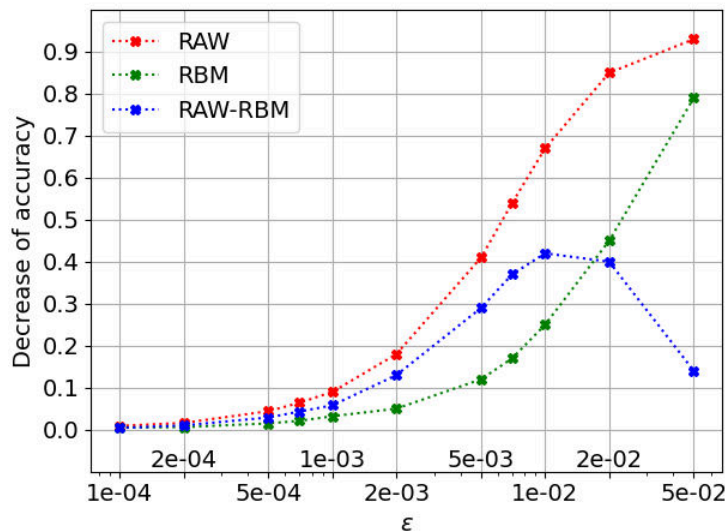


Figure 5.18: Decrease of accuracy on the testing dataset versus epsilon in adversarial attacks.

### 5.3.3 Overall distortion sensitivity of CLBP-RBM preprocessing

**Experiment 13 (Dissertation sensitivity of the CLBP-RBM preprocessing layer).** *This experiment was conducted to verify how the proposed preprocessing method affects classification accuracy for different types of distortion.*

Experiments described in this section were conducted to validate the generalisation ability of the CLBP-RBM method for input images distorted

with different types of image perturbation. As previously shown, adding the CLBP-RBM layer may improve the overall recognition accuracy of CNN classifiers when the input images are perturbed. To investigate the basic abilities of denoising by the CLBP-RBM layer a simple random noise was applied to the input images, then the more complex adversarial distortion was tested. However, in a wider context, robust image processing systems should be insensitive to many types of distortion as their occurrence is a condition not known prior in many cases. Especially, applications that support control loops such as vision systems in robots and industrial machines should be not vulnerable to noise not to cause invalid control signals. However, many sorts of perturbation can be easily simulated with mathematical models [189, 190], thus validation of the sensitivity is relatively easy. This section presents the results of testing the sensitivity of the following types of perturbations that may occur in common processing systems:

- Gaussian noise,
- ISO noise,
- JPEG compression noise,
- Gaussian blur,
- optical distortion (barrel and pincushion),
- multiplicative noise.

The *Gaussian noise* may occur in standard image processing pipelines due to poor illumination, unusual thermal conditions, or may propagate from electronic circuits and has a stochastic nature. This can be formalised by modeling the probability density function as a normal distribution with,  $\mu$  denoting a mean and  $\sigma$  denoting a standard deviation. The *ISO noise* is also stochastic and mainly caused by a too-sensitive camera, it can be modeled by transforming an image to an HSV colourspace and adding noise with a Poisson distribution to a hue channel and normally distributed noise to a saturation channel. The *JPEG noise* is deterministic and occurs in the encode-decode process of compression as it is a lossy compression. In practice, it may happen in image transmission systems. The *Gaussian blur* is a deterministic distortion that can be used to simulate camera defocusing or motion and so unsharp objects. Mathematically it can be achieved by convolving an image with a squared kernel of normally distributed weights. The *optical distortion* is a geometrical transformation that simulates distortion caused by the shape of a camera lens. The *multiplicative noise* is stochastic

and refers to a type of stochastic noise that gets multiplied into the input image. This may propagate from different sources, it can be either caused by a partially broken sensor or electronic circuits distorting the signals. Some example images with applied noise are presented in Table 5.11. As previously mentioned, modern classification pipelines should be insensitive to these types of perturbation as they may occur independently in a variety of scenarios, especially in systems with limited computing resources where the hardware image pre-processing is not complex, and hence does not handle the distortions itself.

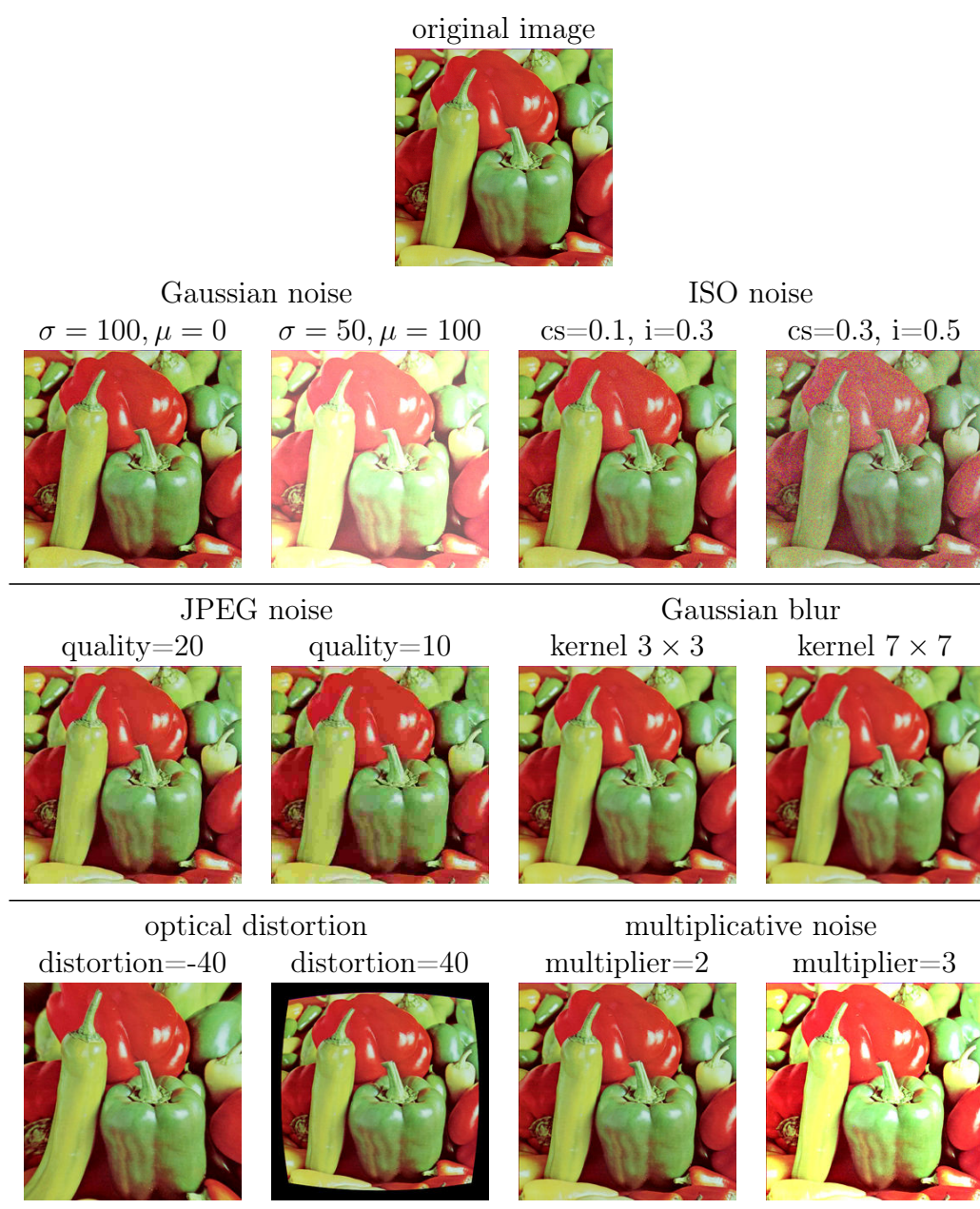


Table 5.11: Example noised images with parameters used for the experiments ("cs" refers to colour shift in ISO noise, "i" refers to intensity). Images were generated with the use of Albumentations library [191] and the parameters of distortions refer to its settings.

The experiments were conducted with the use of the STL-10 dataset and are focused on comparing the classification accuracy of pipelines with and

without the CLBP-RBM preprocessing. The distortions were applied to testing images with the use of the Albumentations library [191] and the further values of parameters refer to settings in this library. As networks without the additional layer achieved lower accuracy, the results of experiments show only the decrease of accuracy in both cases in the following way:

$$d = \frac{\text{accuracy without distortion} - \text{accuracy with distortion}}{\text{accuracy without distortion}}.$$

For each type of perturbation and each set of parameters,  $d$  was computed for the architecture with and without the preprocessing. Results are presented in Tables 5.13, 5.14 and Figures 5.20, 5.19, 5.21. These show the dependencies between  $d$  and the values of parameters of noises. For the clarity of view, three-dimensional graphs present only the difference between the decrease in accuracy for the network with and without the CLBP-RBM layer. To verify the overall quality of distorted image classification,  $d$  values were averaged for each type of distortion, and a new metrics that presents how much the CLBP-RBM layer improves the accuracy was computed:

$$o = \frac{d_{RAW} - d_{RBM}}{d_{RBM} + d_{RAW}}.$$

Values of  $o$  obtained for the considered cases are presented in Table 5.12. We can conclude that the adding the preprocessing reduces the decrease of accuracy in most cases, hence this model is less noise sensitive than a similar one without the CLBP-RBM layer. The only exception is a compression noise where the network processing raw images were almost completely insensitive to this type of perturbation. The compression noise introduces additional regional quantization to the signal. As a result, the binary descriptors become highly sensitive to this type of distortion because detecting local features in a compressed image becomes challenging. The other conclusion is that the highest improvement achieved with the preprocessing was for the *optical distortion*, however mostly for "barrel" perturbation. For the *Gaussian noise* the decrease of accuracy was significant only when the *mean* parameter was different from zero, for its higher values, the network utilising the CLBP-RBM preprocessing was less sensitive to the noise. To visualise this dependency, some examples are shown in Table 5.15. One may generalise the conclusion for other types of perturbations that the use of the proposed denoising method resulted in lower noise sensitivity and this effect is more distinct for higher parameters of the distortion.

**Result of experiment 13.** *The HS-CNN network was less sensitive on different types of perturbation except JPEG compression noise when compared to the network without preprocessing.*



	Gaussian noise	ISO noise	Optical distortion	Gaussian blur	Compression noise	Multi-plicative noise
<b>o</b>	0.378	0.043	0.85	0.35	-0.83	0.274

Table 5.12: Accuracy improvement of the CLBP-RBM preprocessing method for different types of noises.

		mean				
		-100	-50	0	50	100
std	<b>10</b>	0.119/0.279	0.017/0.088	0/0	0.019/0.023	0.09/0.153
	<b>25</b>	0.118/0.281	0.022/0.088	0/0.001	0.016/0.023	0.087/0.155
	<b>50</b>	0.116/0.281	0.018/0.09	0/0	0.015/0.022	0.089/0.155
	<b>75</b>	0.122/0.283	0.023/0.091	0/0	0.017/0.024	0.086/0.155
	<b>100</b>	0.117/0.284	0.015/0.093	0.004/0.004	0.018/0.023	0.093/0.157
	<b>125</b>	0.12/0.287	0.023/0.091	0.002/0.005	0.021/0.026	0.092/0.158
	<b>150</b>	0.112/0.288	0.021/0.095	0.01/0.006	0.026/0.026	0.095/0.157
	<b>200</b>	0.119/0.289	0.024/0.1	0.009/0.003	0.027/0.029	0.1/0.163

Table 5.13: Decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different Gauss noise parameters, results are presented as "RBM/RAW".

		colour shift				
		0.3	0.4	0.5	0.6	0.7
intensity	<b>0.1</b>	0/0	0/0	0/0	0/0	0.003/0.001
	<b>0.2</b>	0/0	0.006/0	0/0	0.005/0.004	0.004/0.004
	<b>0.3</b>	0.007/0	0/0.003	0.009/0.006	0.006/0.009	0.016/0.016
	<b>0.4</b>	0.007/0.006	0.006/0.005	0.009/0.016	0.019/0.018	0.021/0.024
	<b>0.5</b>	0.008/0.007	0.01/0.018	0.021/0.024	0.026/0.03	0.037/0.032
	<b>0.6</b>	0.019/0.014	0.022/0.026	0.024/0.032	0.04/0.036	0.043/0.04
	<b>0.7</b>	0.022/0.024	0.035/0.037	0.029/0.038	0.045/0.04	0.042/0.046
	<b>0.8</b>	0.033/0.036	0.043/0.048	0.034/0.048	0.055/0.055	0.051/0.056
	<b>0.9</b>	0.036/0.045	0.047/0.05	0.052/0.058	0.053/0.062	0.047/0.053
	<b>1</b>	0.045/0.061	0.055/0.06	0.06/0.067	0.057/0.068	0.057/0.057

Table 5.14: Decrease of accuracy in a network with and without the CLBP-RBM preprocessing depending on different ISO noise parameters, results are presented as "RBM/RAW".

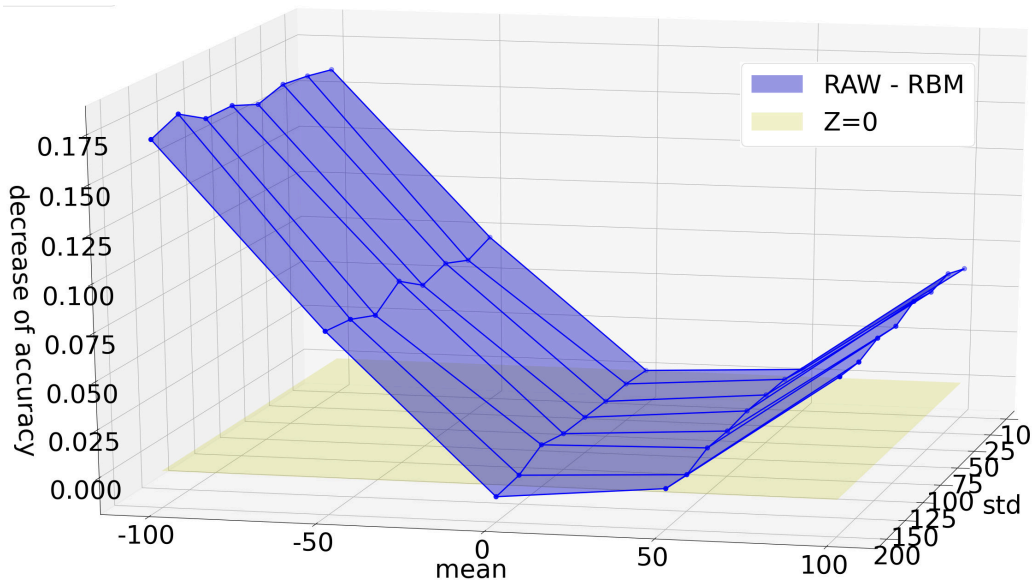


Figure 5.19: Difference in decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different Gauss noise parameters.

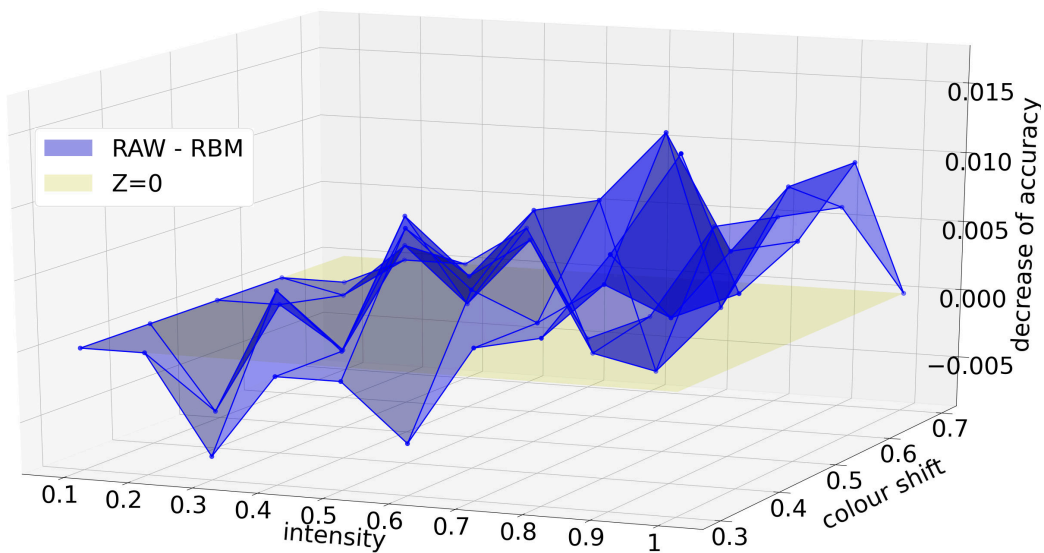


Figure 5.20: Difference in decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different ISO noise parameters.

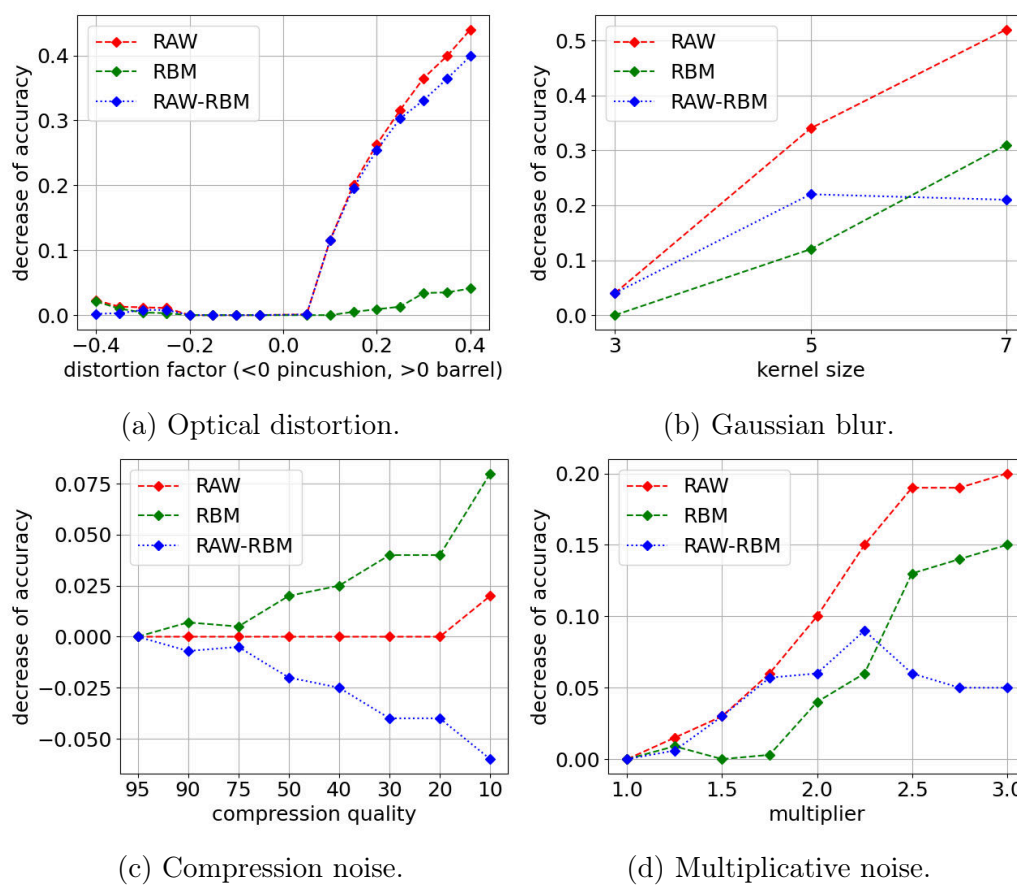


Figure 5.21: Decrease of accuracy in networks with and without the CLBP-RBM preprocessing depending on different distortion parameters, blue lines denote the difference in decrease of accuracy.




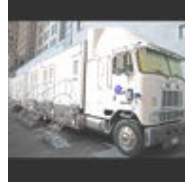

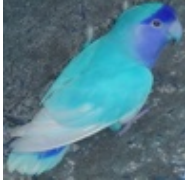







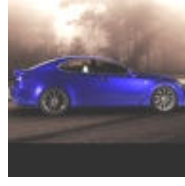

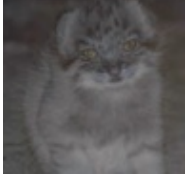
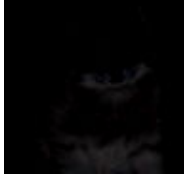



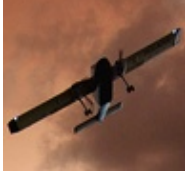

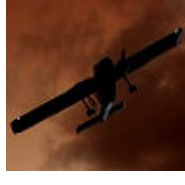
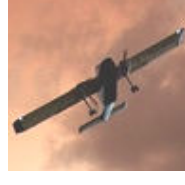
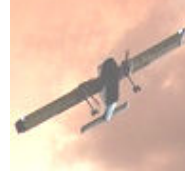
original	mean = -100	mean = -50	mean = 50	mean = 100
				
truck(0.99) truck(0.98)	truck(0.97) ship(0.54)	truck(0.98) truck(0.93)	truck(0.99) truck(0.86)	truck(0.76) ship(0.73)
				
bird(0.86) bird(0.79)	bird(0.75) car(0.85)	bird(0.84) bird(0.50)	bird(0.79) bird(0.38)	bird(0.50) deer(0.40)
				
car(0.99) car(0.92)	airplane(0.85) truck(0.90)	airplane(0.97) truck(0.94)	car(0.92) airplane(0.67)	car(0.92) airplane(0.94)
				
cat(0.88) cat(0.68)	cat(0.42) deer(0.40)	cat(0.88) cat(0.84)	cat(0.82) deer(0.50)	cat(0.69) deer(0.64)
				
airplane(0.70) airplane(0.99)	bird(0.32) bird(0.62)	airplane(0.65) airplane(0.89)	airplane(0.71) airplane(0.99)	bird(0.49) airplane(0.98)

Table 5.15: Example images distorted with *Gaussian noise* and predictions from network with and without CLPB-RBM preprocessing. Green text denotes predictions with preprocessing, red without, predicted probability is given in parentheses.

## 5.4 IMAGE CLASSIFICATION WITH CONTRASTIVE DIVERGENCE AS A FEATURE SPACE

### 5.4.1 Binary descriptors aggregation

**Experiment 14 (Image classification with CD-KNN architecture).** *This experiment was designed to validate the classification ability of CD-KNN architecture.*

The experiments described in this section relate to CD feature space and binary descriptors aggregation which were described in section 4.2. Caltech101 was used as an image dataset, and samples from it were split to make use of unsupervised learning, thus a predefined number of images have been used to optimise RBM parameters, and a subset of them to train a KNN classifier. The processing pipeline is shown in Figure 4.1. The first experiment was to validate which type of binary descriptor performed best in this classification architecture. Results for four types of descriptors are shown in Table 5.16, where LBP16 refers to an LBP descriptor with a radius equal to 2.25.

	100/30	100/50	200/30	200/50
BRISK	0.598;0.603	<b>0.605;0.613</b>	0.604;0.603	0.604;0.612
FREAK	0.613;0.575	0.618;0.580	0.614;0.571	0.618;0.574
LBP8	0.583;0.560	0.606;0.594	0.591;0.583	0.592;0.574
LBP16	0.581;0.550	0.581;0.551	0.582;0.554	0.584;0.558

Table 5.16: The evaluation of a CD-based KNN matching procedure, results are presented as "accuracy/mean Average Precision", the first row denotes "number of images for RBM training/number of images for KNN training", the best result is highlighted with a bold font.

The classification quality was similar for different types of descriptors. BRISK achieved the best performance, however, LBP8 worked well in this case, since its accuracy was close to BRISK, however LBP8 was much faster (see Table 2.2).

**Result of experiment 14.** *CD-KNN architecture can be used for classification purposes and achieved the greatest accuracy for BRISK descriptor.*

### 5.4.2 CD as an entry for CNN

---

**Experiment 15 (Image classification with CD-CNN architecture).**  
*This experiment was designed to validate classification ability of CD-CNN architecture.*

Another method presented in 4.2 was to use CD features as an entry to a Convolutional Neural Network. To validate the capabilities of applying this method for image classification similar experiments were conducted as in the case of aggregation. The first experiment was conducted to validate what type of descriptor worked best in this type of pipeline. This case also took the colour into account. This was done in CLBP as described in 2.5 and with LBP8\_rgb which was a 24-bits descriptor formed by concatenating three LBP8 descriptors each for one colour channel. The results are shown in Table 5.17.

Another experiment showed how adding a CD features layer affected the overall classification accuracy, the additional preprocessing was turned on and off as were the descriptor layers, and the results are shown in Table 5.18. Figure 5.22 presents a comparison of the processing performance between the proposed preprocessing method and some other commonly known deep neural network architectures. Additionally, an experiment that compares how modifying the size of an input image affected the network performance was conducted. The results in Table 5.19 show that with the use of the preprocessing it is possible to reduce the network complexity by changing the input size without decreasing the generalisation ability significantly, which may be a interesting property when applied in systems with limited computing resources.

The final conclusion is that the proposed method can be successfully applied for an image classification problem, and the complexity of the network can be made relatively low due to the initial preprocessing. However, this type of processing should be considered weaker when compared to the approach using hidden space for feature extraction, this is mainly because CD space is bigger than latent space, hence there are more parameters of CNN that have to be optimised. On the other hand, CD feature space has an interesting property of descriptors aggregation. The lack of efficient was indicated in Section 2.4.4 as potential weakness of binary descriptor in general.

**Result of experiment 15.** *CD-CNN architecture can be used for classification purposes and achieved the greatest accuracy for an LBP8 descriptor.*

descriptor	validation accuracy	validation top5	training accuracy
LBP8	0.68	0.82	0.99
LBP16	0.65	0.87	0.98
LBP8_rgb	0.42	0.70	0.81
CLBP	<b>0.72</b>	0.85	0.99
BRISK	0.51	0.72	0.98

Table 5.17: The evaluation of the accuracy of CD-RBM as an entry for DNN

BD type	BD + RBM	BD	None (greyscale)	None (RGB)
LBP8	0.68	0.63	0.54	0.62
CLBP	0.72	0.65	0.54	0.62

Table 5.18: The evaluation of network accuracy with RBM preprocessing and without it, "BD" refers to a binary descriptors layer.

size of image	accuracy	time reduction [%]
256x256	0.74	0
181x181	0.73	50
128x128	0.71	70
90x90	0.72	77
63x63	0.66	84
44x44	0.68	85

Table 5.19: Validation accuracy depending on the size of the input image for an RBM with 10 hidden neurons and a CLBP descriptor as an input.

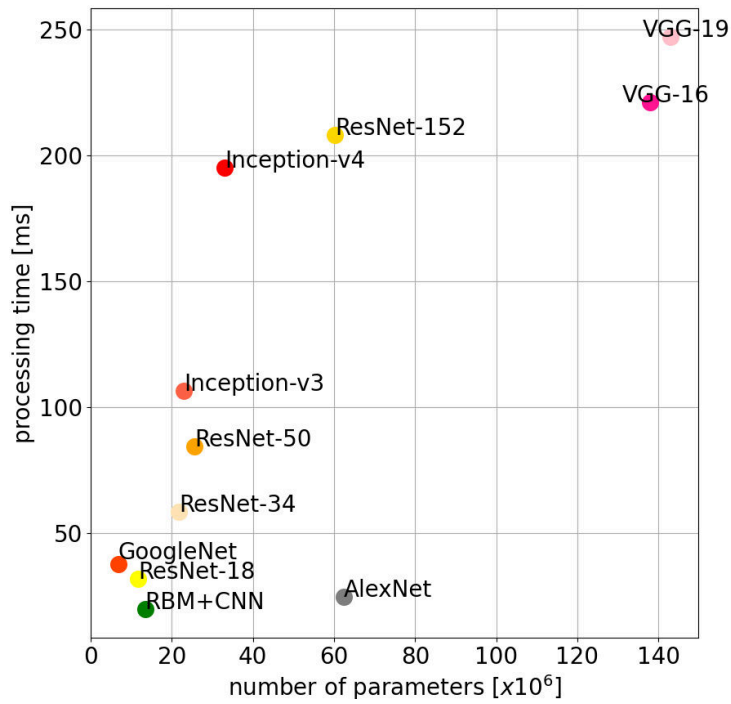


Figure 5.22: Comparison of the processing time and the number of parameters between the RBM+CNN and some commonly known deep neural architectures.



---

## MOBILE ROBOT APPLICATION

---

The previous chapters introduced and described in detail all the concepts related to the CLBP-RBM preprocessing. It has also been stated that this may be potentially applied in robotics as it addresses some of the commonly known concerns in systems with limited resources or data availability. This chapter presents an exemplification of applying the proposed preprocessing in a mobile robot.

### 6.1 APPLICATION GOALS AND EXPERIMENTAL SETUP

---

The application implements a basic visual perception module and it is focused on validating whether or not the preprocessing proposed in the previous part of this dissertation may be successfully transferred to an embedded system, and how it can improve the ability of the robot to interact with the environment. The application under consideration is not designed to support a complex navigation/control problem since this is beyond the scope of this study. Instead, it can be seen as an exemplification of a perception module based on the CLBP-RBM layer, which can be further utilised for integration with control loops.

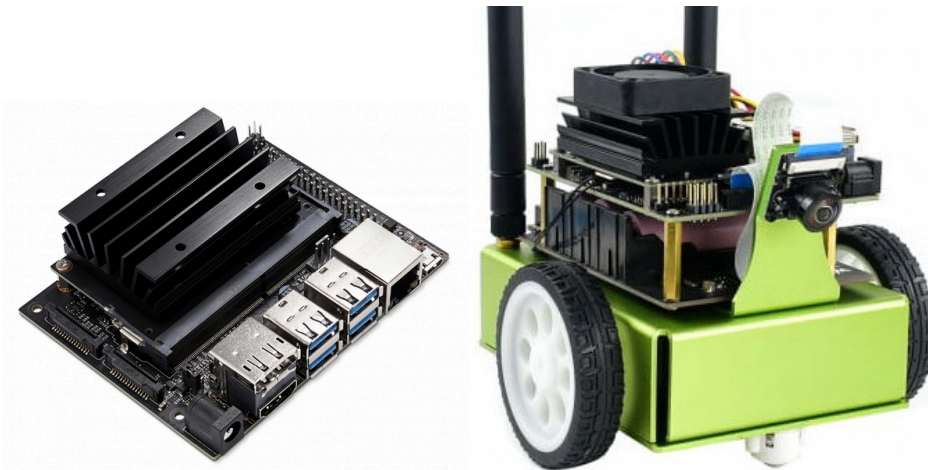
Before starting an analysis of the application it is worth defining the hardware requirements. As the implementations of a CLBP and an RBM are focused for working on GPUs to utilise their highly parallel nature, the system should be equipped with such a computational unit. However, it does not have to be very efficient which meets the potential requirement of running the application on small, inexpensive devices. The obvious need is also access to at least one video sensor, and a WiFi module to allow faster development and testing. A computing unit in this case will be utilised to detect objects around the robot and to set the desired position of the robot based on visual feedback. Hence, the physical parameters of a robot are not significant as they do not affect the classification procedure. Taking these requirements

into account, one may consider the JetBot [192] development kit as a good solution. This is based on the Jetson Nano module [193]. It provides an easy method of building a small two-wheeled mobile robot that can be supervised by the Jetson Nano, which runs the Linux operating system, offering a high-level of versatility to the programming of the robot. Figure 6.1 presents the robot and the board, the hardware details of Jetson Nano are as follows:

- CPU - quad-core ARM A57, 1.43GHz,
- GPU - 128-core Maxwell Architecture with 2GB VRAM,
- RAM - 4GB 64-bit,
- Camera Interface - 2x MIPI CSI-2,
- Peripherals - HDMI, 4x USB, Ethernet,
- Mechanical - 69mm x 45mm.

JetBot provides main electro-mechanical and electronic components, that can be directly connected to the board:

- batteries supply module,
- camera - Sony IMX219,
- small OLED display for debugging,
- Bluetooth and WiFi module,
- two DC motors with gears,
- wheels and cover.



(a) NVIDIA Jetson Nano.

(b) JetBot Nano.

Figure 6.1: Hardware used for the mobile robot application.

## 6.2 THE ARCHITECTURE OF THE APPLICATION

From a very general point of view, the application can be split into two modules. The first is the program that runs on the robot. The second is a simple program that connects with the robot, gathers the results and passes some control commands, it can be run on any device running Linux, Windows, or macOS, it will be referred to as *receiver* since it plays rather a passive role. Both are written in python, they use ZeroMQ [194] for communication purposes which has been chosen because of its simplicity and high performance. The receiver app utilises OpenCV for some basic image processing purposes and tkinter [195] to provide a graphical user interface. Tkinter has been chosen because it is cross-platform and easy to use. It allows users to observe what the robot camera input in real-time and also to run some control commands for motion and for recognition. Analogously, the robot program gathers and sends the images from the camera to the receiver, enables the prediction process on demand and sends results. The bottleneck for the communication in real-time is frames passing. Sending raw data may cause delays as a single frame is relatively big (1.17MB for 640×480 RGB image) and there may be obstacles between the robot and the receiver app that may slow down wireless communication. Hence, the robot decodes frames with JPG compression, and the receiver does encoding which allows sending 640×480 frames with a speed of 30FPS. A very general diagram of the application is shown in Figure 6.2.

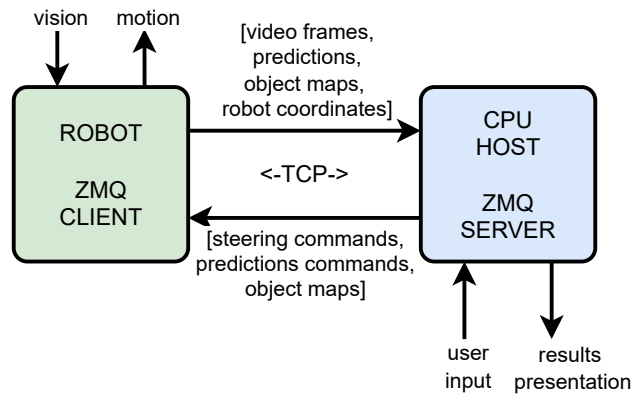


Figure 6.2: General diagram of the experimental application.

The robot application is relatively complex, since it needs to communicate with the receiver, support basic motion control tasks, gather data from the camera sensor, and process the video with a neural network. Therefore

the program is implemented as a multithreaded process to provide the highest possible performance. The simplified architecture of this is schematically shown in Figure 6.3. From a practical point of view, the most important aspect is that one thread is responsible for capturing camera frames and sending them compressed to the receiver, another one performs an estimation of the robot position, the main thread does the processing with CLBP-RBM and neural network and performs some actions based on data gathered from the other threads. The receiver application is significantly less complex, however to work in real-time it is also split into two threads. The first one receives and sends the data, and the second presents the image frames and predictions received from the robot. An example screenshot from the application working is shown in Figure 6.4. We can observe that this is split into two parts. The first, shows video frames with the actual prediction at the top (red rectangle) and coordinates of the robot (yellow rectangle). The second shows buttons for sending user input, robot controls (green rectangle), advanced recognition requests (pink rectangle), and saving frames (blue rectangle).

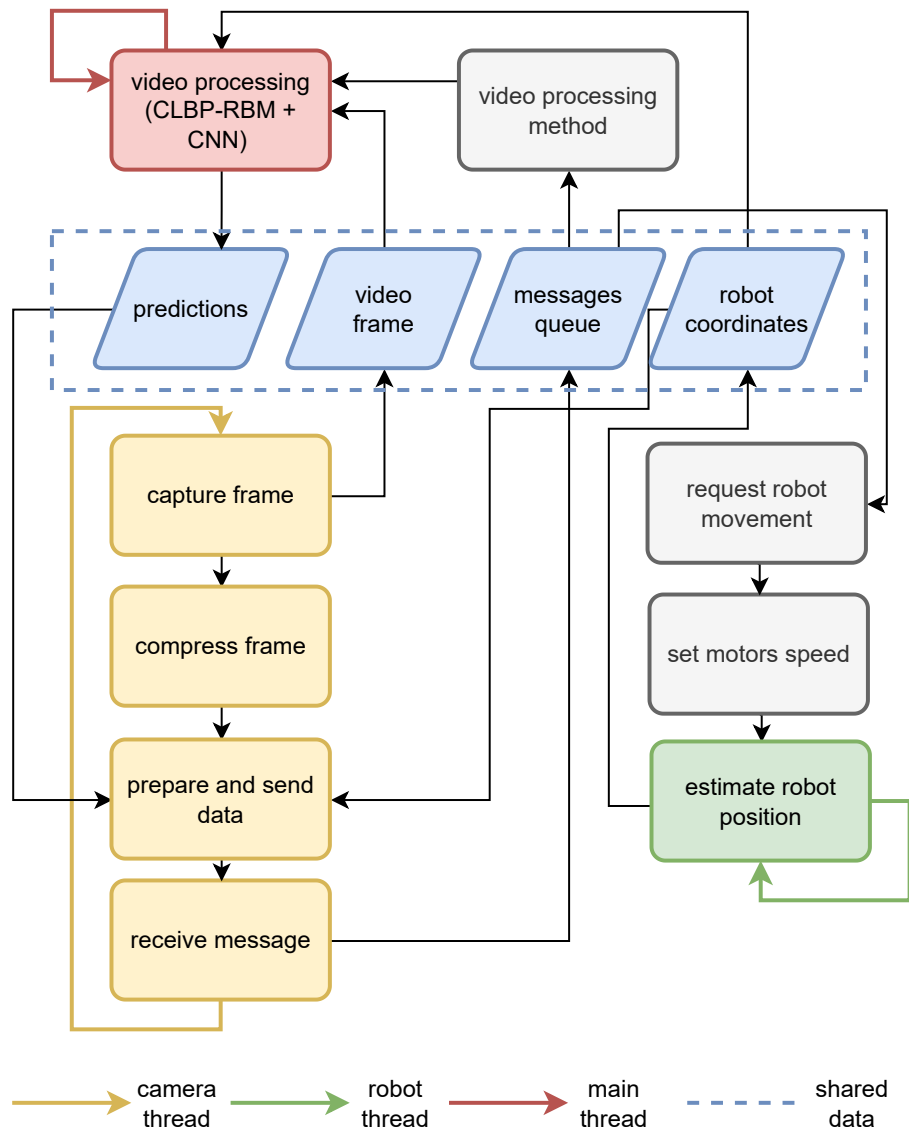


Figure 6.3: Architecture of the robot's program.

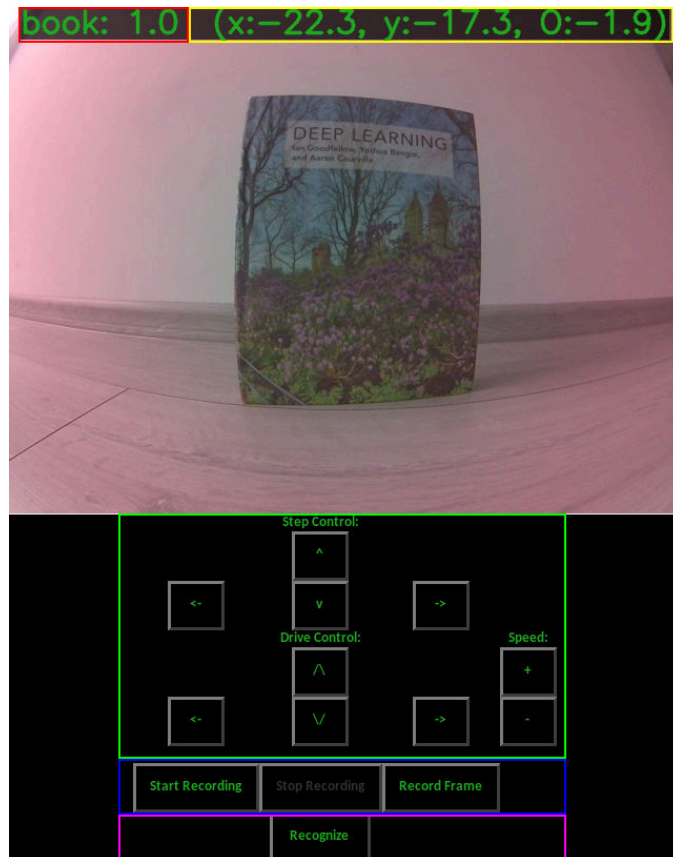


Figure 6.4: Screenshot from the receiver application. Coordinates are shown in centimetres, "O" stands for  $\theta$ .

Here, it is worth mentioning, that besides the visualisation purposes, the application was designed to automatically save the frames that are currently processed by the robot. This is useful for training a robot's neural network. In short, it is possible to enable automatic frames saving, run some random robot's motion and then the saved frames may be utilised to train an RBM in an unsupervised fashion. A subset of them may then be manually labelled and used to train a CNN. This is formally the main concept of applying CLBP-RBM in mobile robots because it is possible to train the preprocessing without significant effort from an engineer and train a CNN with the limited amount of data needed to provide good generalisation ability.

## 6.3 MOBILE ROBOT KINEMATICS

Despite the considered application not focusing on planning the robot's motion, an estimation of the robot's position may be useful for interpretation of the gathered data. The equations of a mobile robot's kinematics are commonly known [196], and can also be applied for this case.

The robot moves in planar coordinates  $x, y$  with an orientation angle  $\theta$ , assuming it starts its motion at point  $x_0, y_0, \theta_0$  it is possible to determine its position  $x_t, y_t, \theta_t$ , at time  $t$ . The geometric parameters needed to estimate the position are only robot width -  $w$ , and wheel diameter  $d$ . The velocities known at each time  $t$  are the wheels' angular velocities  $\omega_{t_l}$  for the left wheel,  $\omega_{t_r}$  for the right wheel, which can be further transformed to the robot's angular velocity  $\dot{\theta}_t$ , and linear velocity  $v_t$ , which can be split into velocities along  $O_x, O_y$  axes -  $v_{t_x}, v_{t_y}$ , this is visualised in Figure 6.5.

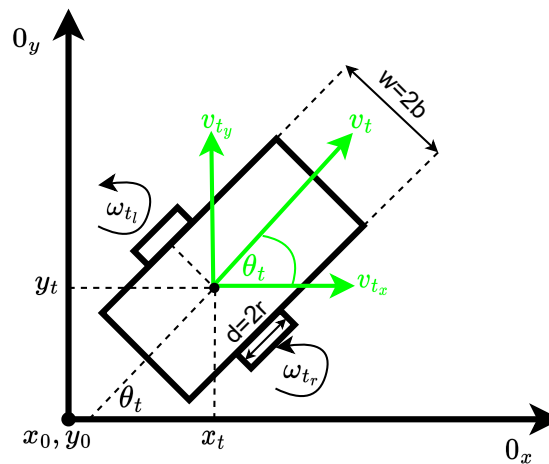


Figure 6.5: Diagram of a mobile robot in its local coordinates.

For such a defined case, one can write the following matrix-vector equation to determine robot velocities:

$$\begin{bmatrix} v_t \\ \dot{\theta}_t \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2b} & -\frac{r}{2b} \end{bmatrix} \begin{bmatrix} \omega_{t_l} \\ \omega_{t_r} \end{bmatrix}, \quad (6.1)$$

taking into account that  $v_{t_x} = \cos(\theta_t)$  and  $v_{t_y} = \sin(\theta_t)$  and rewriting to a

system of equations, one can write:

$$\begin{aligned}\dot{\theta}_t &= \frac{d}{w}(\omega_{t_r} - \omega_{t_l}) \\ v_{t_x} &= \frac{r}{2} \cdot \cos(\theta_t) \cdot (\omega_{t_r} + \omega_{t_l}) \\ v_{t_y} &= \frac{r}{2} \cdot \sin(\theta_t) \cdot (\omega_{t_r} + \omega_{t_l}),\end{aligned}\tag{6.2}$$

and finally, it possible to write the equations for robot coordinates:

$$\begin{aligned}\theta_t &= \theta_0 + \frac{d}{w} \int_0^t (\omega_{\tau_r} - \omega_{\tau_l}) d\tau \\ x_t &= x_0 + \frac{r}{2} \int_0^t \cos(\theta_\tau) \cdot (\omega_{\tau_r} + \omega_{\tau_l}) d\tau \\ y_t &= y_0 + \frac{r}{2} \int_0^t \sin(\theta_\tau) \cdot (\omega_{\tau_r} + \omega_{\tau_l}) d\tau.\end{aligned}\tag{6.3}$$

The last step needed for an implementation is a discretisation of these equations:

$$\begin{aligned}\theta_t &= \theta_0 + \frac{d}{w} \sum_{\tau=0}^t (\omega_{\tau_r} - \omega_{\tau_l}) \cdot d\tau \\ x_t &= x_0 + \frac{r}{2} \sum_{\tau=0}^t \cos(\theta_\tau) \cdot (\omega_{\tau_r} + \omega_{\tau_l}) \cdot d\tau \\ y_t &= y_0 + \frac{r}{2} \sum_{\tau=0}^t \sin(\theta_\tau) \cdot (\omega_{\tau_r} + \omega_{\tau_l}) \cdot d\tau,\end{aligned}\tag{6.4}$$

where  $d\tau$  is a sampling step.

### 6.3.1 Identification of mobile robot parameters

---

The geometric parameters of the robot used for experiments are:  $w = 0.12\text{m}$  and  $d = 0.0365\text{m}$ , the only concern is to determine kinematic control:  $\omega_{t_x}, \omega_{t_y}$ . One of the commonly used methods to do that is to use encoders, which allows measuring wheel angle and therefore its angular velocity by differentiation. However, the robot used in experiments was not equipped with encoders, so the method to estimate wheels' velocities was based on motors' control inputs. The idea is relatively simple, we can control robot motor speed in open-loop by setting a normalised PWM value ( $k \in \langle -1; 1 \rangle$ ) to its power amplifier. We we can experimentally identify characteristics  $\omega(k)$ , which

---



describes the steady state velocity for a given PWM input determined by  $k$ . The results of the experiment are shown in Figure 6.6. The dependency is clearly linear which makes the estimation simpler. For  $k \in (-0.075; 0.075)$  a dead-zone is detected - the robot does not move. Moreover, there is a slightly different characteristic for negative  $x$  values than for positive  $x$  values, hence the estimated velocity of wheels can be written as follows:

$$\omega(k) = \begin{cases} 51.39k + 1.87 & \text{for } k \leq -0.075 \\ 51.32k - 2.03 & \text{for } k \geq 0.075 \\ 0 & \text{for } k \in (-0.075; 0.075) \end{cases} \quad (6.5)$$

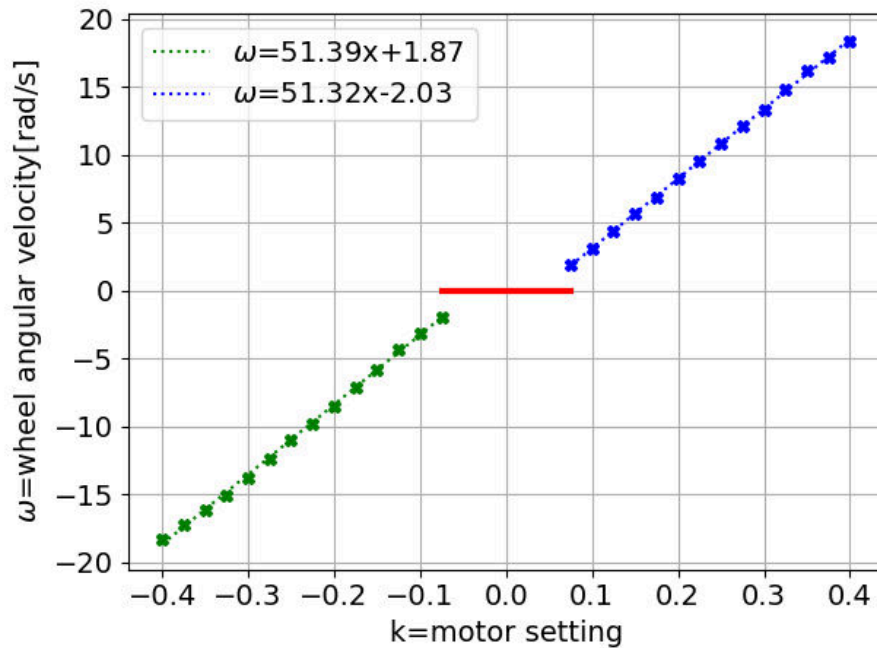


Figure 6.6: JetBot's wheel velocity estimation.

## 6.4 RBM FOR ROBOT ROTATION ANGLE DETECTION

**Experiment 16 (Mobile robot rotational movement based on CLBP-RBM probabilities histogram visual feedback).** *This experiment was designed to verify if the mobile robot could perform a single*

*rotational movement based on CLBP-RBM visual feedback only. This verified the hypothesis that the comparison of probabilities histograms may be used to measure the similarity between set of images in a real environment.*

The application focused mainly on detecting objects in the robot's environment. Basic usability was to set up the robot orientation in such a way to be positioned along a desired object's axis. The high-level algorithm to perform this task will be explained later in detail, however, it was expected that the robot would perform a single rotation to infer the best angle when the visibility of a given object is the highest. It would not be possible without  $\theta_t$  estimation, however, CLBP-RBM could provide a method allowing the detection of a full rotation. The idea is not very complex: since CLBP-RBM provides a method for measuring the similarity between two frames, it is possible to detect if during the rotational motion a robot processes the same (or very similar) frames as at the beginning. In the beginning, the vision system gathers statistics from an environment and remembers its histogram as a reference, then during a rotational motion robot stops when the currently processed histogram is similar to the reference, which means if the distance between these histograms is less than an average distance to frames in a reference environment.

The experiment to validate if this technique can be used for the desired task was focused on testing how well the robot performed a single rotation when compared to a method that relied on robot kinematics (6.4), and a velocities estimation (6.5). For measurements, a dedicated vision system which allowed the measurement the angle deviation of a robot's axis from the desired axis was used. This method is explained in detail in the appendix C, in short, it allowed automatic measurement of the angle by using special markers on the robot and on the ground and taking photos of particular configurations. This technique provided sufficient precision for that task and was convenient to use because of its portability and speed for a large number of testing points. The experiments were conducted for four rotational velocities and different lighting conditions. The results are shown in Figure 6.7. For the case of CLBP-RBM, the angle deviations were negative or positive, which meant the robot had performed more than one full rotation or less, in the case of angle estimation, errors were negative, so the robot performed less than one rotation. The errors did not vary significantly for different  $\omega$ , however they were lower for the CLBP-RBM method when compared to the angle estimation method. In conclusion, the proposed method of applying CLBP-RBM for indicating the stop point for the robot performing a single rotation was sufficient to be used for other experiments that relied on localising objects around the robot. It is not a technique that would allow setting an

orientation of the robot precisely, however, it showed that the proposed preprocessing may have been useful for practical reasons in this dissertation as small orientation errors are acceptable in terms of testing the robot's visual perception.

**Result of experiment 16.** *With the help of histograms comparison, the mobile robot could perform a single rotational movement with higher accuracy than based on wheels velocities estimation.*

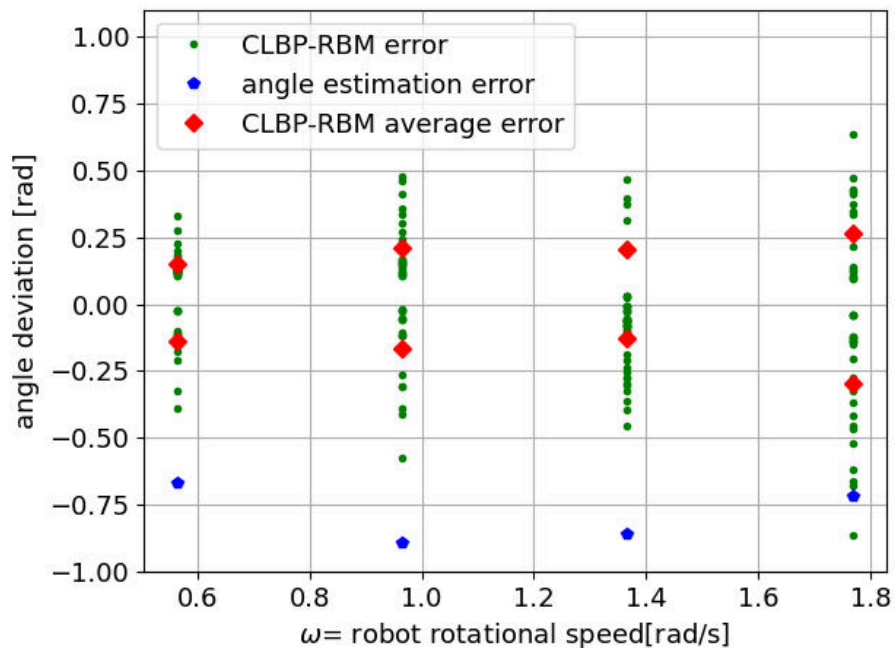


Figure 6.7: Mobile robot error on performing one rotation with the use of CLBP-RBM and angle estimation. Negative value of the deviation denotes that the robot performed less than one rotation, positive value denotes more than one rotation. The error from visual feedback should be interpreted as random while the errors from angle estimation are systematic.

## 6.5 CLBP-RBM PREPROCESSING FOR A CNN IN MOBILE ROBOT VISION SYSTEM

**Experiment 17 (Objects classification by the mobile robot with HS-CNN processing pipeline.).** *This experiment was designed to verify if the previously discussed classification pipeline could be successfully applied*

*in the robot's object recognition system and how it performed when compared to a raw pixel input.*

To test the recognition capabilities, three objects were chosen that robot could categorise based on the camera input. They were:

- a mini air conditioner,
- a book,
- a camera,

and are shown in Table 6.1. It is worth noticing that these objects were random and were not associated with a specific task defined for the robot. Instead, they were analysed in this study as generalised examples of classification cases. These were selected to be relatively complex in terms of recognition problem because did not have outstanding visual features when compared to the environment used. The environment for the experiments was a regular house room.

An RBM was trained based on the images gathered by the robot camera, which was no burdensome as it was automatised by the application described earlier. After collecting a number of frames - around 20000 in that case, some of them were labelled into four categories (three objects plus a negative category) - around 500 per category. The RBM was trained on all the images, then a simple CNN network was trained on the labelled samples. As the system has some limitations in terms of performance the network has been designed to be small to ensure the real-time response of the recognition system. In spite of this small CNN size, with the use of CLBP-RBM preprocessing it was possible to achieve 97% of accuracy. The time of processing on the target device was lower than 14ms which ensured real-time processing even for cameras working in 60 frames per second mode. This is achievable also by the network optimisation performed in TensorRT [197], such an optimised network resulted in faster responses without losing the precision of the computations, it is also worth mentioning that the model responses were the same on the JetBot as on the PC where it was being trained.

To show the advantage of using CLBP-RBM preprocessing in that case, the same model was trained on raw pixel data, and the results are summarised in Table 6.2 to show the efficiency on testing dataset and in Figure 6.8 to show the training process on validation dataset. The evaluation metrics with the preprocessing are significantly better which leads to the conclusion that the proposed method may have helped in terms of classification accuracy. The main reason of the improvement was that the number of training samples was

relatively small, and as was proven in the previous chapter, the CLPB-RBM preprocessing played an important role in this case.

The simplest capability of the recognition system was to report what the robot camera observed at that time. The predictions were sent in real time to the user interface; the example screenshots of the application are shown in Table 6.3. The system, in general, classified input images properly as there were not many incorrectly recognised frames. However, it is worth mentioning that the neural network was trained to focus only on the centre of the image because the camera view was wide, hence analysing the entire frame would have confused the model with such small objects being recognised.

**Result of experiment 17.** *The HS-CNN architecture was successfully applied in the mobile robot, achieved real-time processing speed, and performed better than a CNN with raw pixel input.*

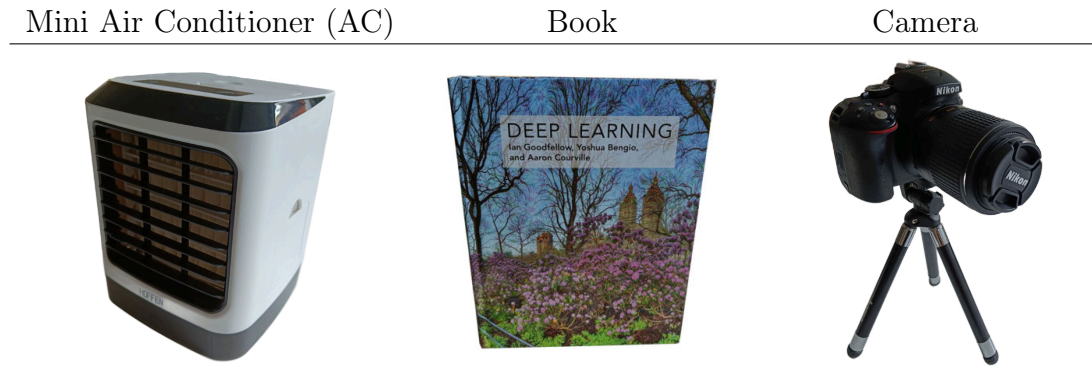


Table 6.1: Objects chosen for robot classification system experiments.

	AC		Book		Camera		Negative	
	RBM	RAW	RBM	RAW	RBM	RAW	RBM	RAW
Accuracy ( $A$ )	0.993	0.974	0.983	0.964	0.996	0.976	0.976	0.922
Precision ( $P$ )	0.991	0.934	0.943	0.931	0.991	0.936	0.975	0.890
Recall ( $R$ )	0.983	0.966	0.991	0.923	0.991	0.991	0.936	0.801
Specificity ( $S$ )	0.997	0.978	0.980	0.977	0.997	0.961	0.991	0.965
$\frac{\sum_{\{A,P,R,S\}} (RBM-RAW)}{4}$	0.028		0.025		0.028		0.075	

Table 6.2: Evaluation of recognition pipeline generalisation ability metrics for each classified category on testing dataset.

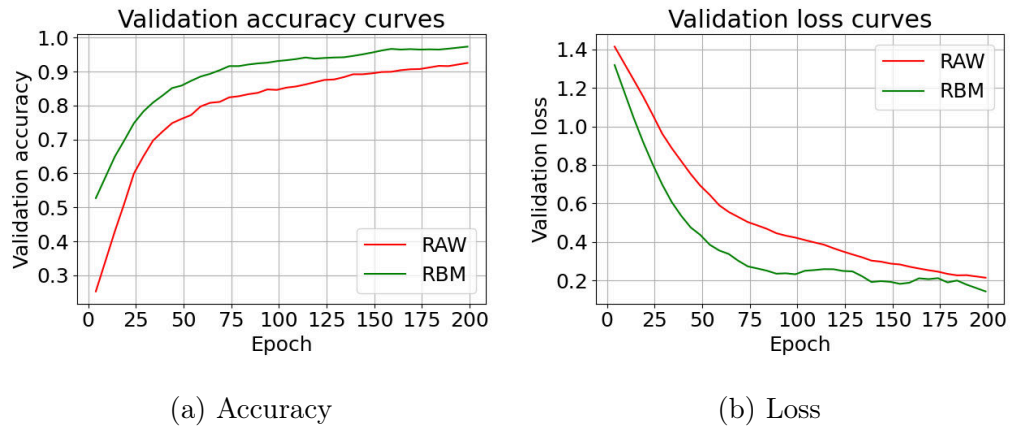


Figure 6.8: Training curves on validation dataset for the mobile robot object classification experiment.

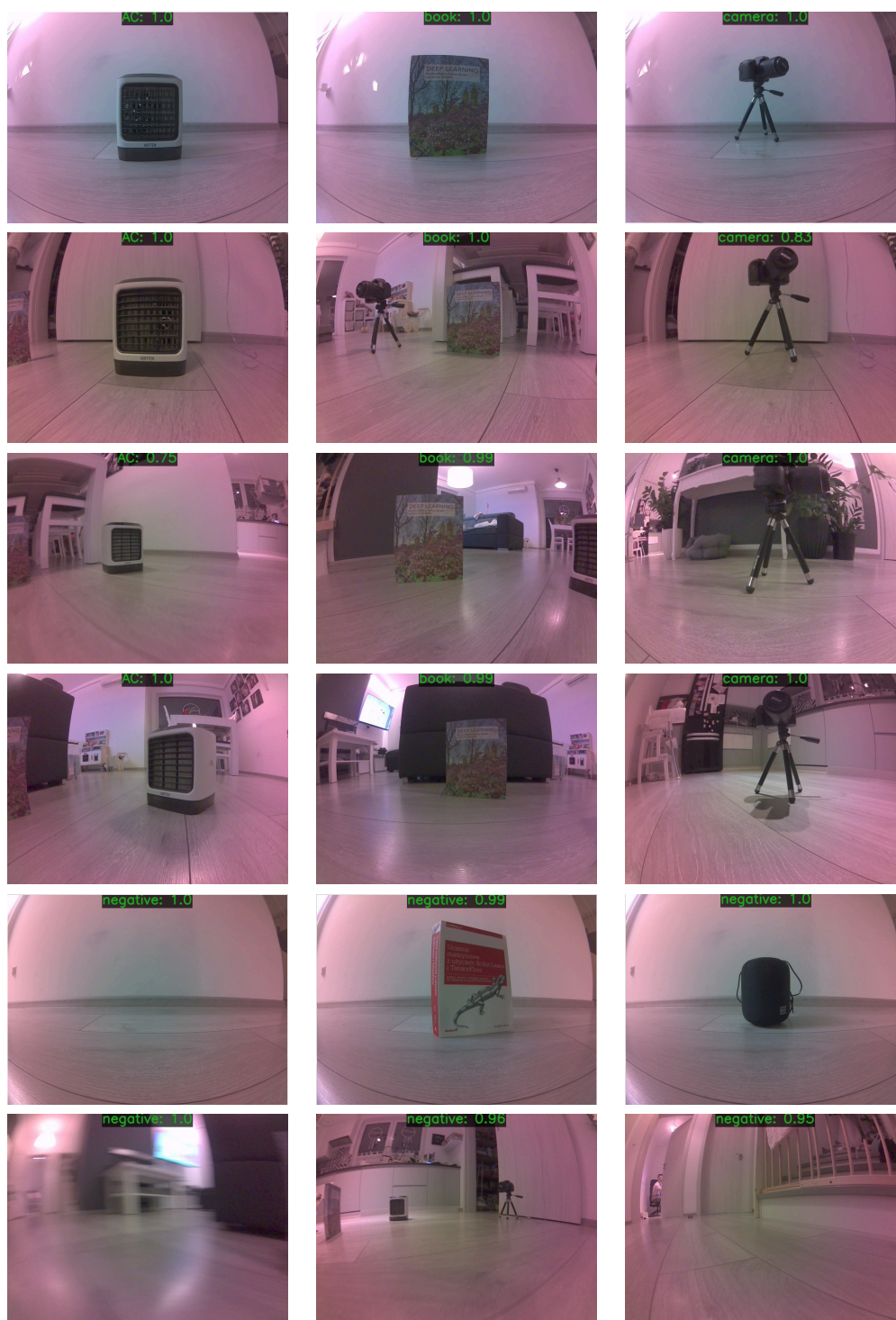


Table 6.3: Example screenshots from robot prediction, coordinates of the robot not displayed for the clarity of view.

**Experiment 18 (Mobile robot positioning along a given object based on visual feedback).** *This experiment was designed to validate if the robot may position along an object being recognised by the classification system.*

Another experiment related to the considered classification pipeline was to recognise where a given object was localised around the robot and to set its position along axis of the detected object. This was mentioned in the previous section. The algorithm could base only on the vision system as because of the CLBP-RBM preprocessing it was possible to make a single rotation to infer where there was the highest probability of occurrence of a given object and then to turn back to it. A simple schematic of this task is shown in Figure 6.9. A pseudocode for this process is shown in the Algorithm 4.

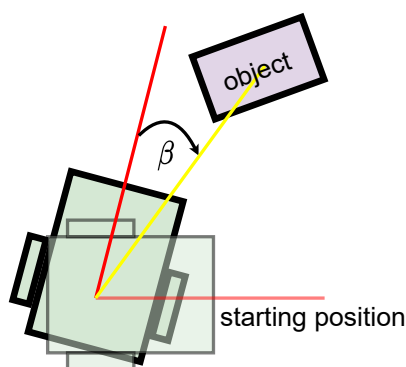


Figure 6.9: Robot orientation task diagram, the robot should be positioned along the yellow line,  $\beta$  is a localisation error.

To validate the accuracy of the orientation system, tests were carried out which measured the angle deviation from the object axis after the positioning command had been sent to the robot. The robot and the objects were spaced out in random positions in the environment and at random angles. The experiments have been performed also in different lighting conditions. The results are presented separately for each of the three objects as shown in Figure 6.10. A precision of less than 0.1 radians may be considered as high, however there is no baseline method that could be used to compare the accuracy. From a practical point of view, we can claim that for all the conducted tests, the robot was positioned correctly to have the desired object in the camera field of view.

**Result of experiment 18.** *The mobile robot could be positioned along a given desired object based on only the visual feedback.*



**Algorithm 4** Orientation of mobile robot along an axis of a given object based on vision system.

---

```
1: procedure get_point_statistics()
2:   referenceDescriptors  $\leftarrow$  [ ];
3:   for step in [-1, -1, 1, 1, 1, 1, -1, -1] do
4:     frame  $\leftarrow$  camera.get_frame();
5:     referenceDescriptors.append(predictor.get_descriptors(frame));
6:     if step = -1 then
7:       robot.step_clockwise();
8:     else
9:       robot.step_counterclockwise();
10:    end if
11:  end for
12:  referenceHistogram  $\leftarrow$  predictor.get_histogram(referenceDescriptors);
13:  frame  $\leftarrow$  camera.get_frame();
14:  histogram  $\leftarrow$  predictor.get_histogram(frame);
15:  threshold  $\leftarrow$  predictor.get_distance(referenceHistogram, histogram);
16:  return referenceHistogram, T
17: end procedure
18:
19: procedure predict_around()
20:   referenceHistogram, threshold = get_point_statistics()
21:   maximumPrediction  $\leftarrow$  0;
22:   bestHistogram  $\leftarrow$  0;
23:   robot.run_clockwise();
24:   while True do
25:     frame  $\leftarrow$  camera.get_frame();
26:     histogram  $\leftarrow$  predictor.get_histogram(frame);
27:     d  $\leftarrow$  predictor.get_distance(referenceHistogram, histogram);
28:     if d < threshold then
29:       break;
30:     end if
31:     probability, label  $\leftarrow$  predictor.predict(frame);
32:     if label = "positive" and probability > maximumPrediction then
33:       bestHistogram  $\leftarrow$  histogram;
34:       maximumPrediction  $\leftarrow$  probability;
35:     end if
36:     step  $\leftarrow$  step + 1;
37:   end while
38:   return bestHistogram, threshold
39: end procedure
```

---

---

```
40:
41: bestHistogram, threshold = predict_around()
42: while True do
43:   frame ← camera.get_frame();
44:   histogram ← predictor.get_histogram(frame);
45:   d ← predictor.get_distance(bestHistogram, histogram);
46:   if d < threshold then
47:     probability, label ← predictor.predict(frame);
48:     if label = "positive" then
49:       break;
50:     end if
51:   end if
52: end while
```

---

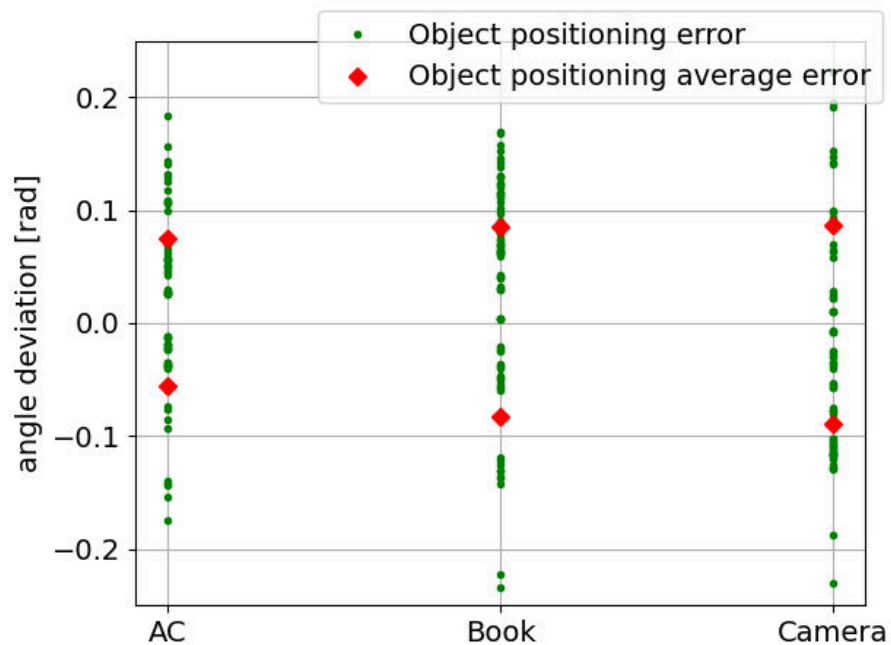


Figure 6.10: Mobile robot angle deviation while positioning along an axis of a given object, 80 measurements for each category were conducted.

**Experiment 19 (Generating objects localisation map in the mobile robot recognition system).** *This experiment was designed to test the capability of orientation of the mobile robot along objects around it.*

Another experiment that was performed with the use of the previously mentioned capabilities was to find angle of localisation of different types of objects around the robot. This was achieved by executing a single circular motion as described in Section 6.4 and gathering the predictions. After this was finished the coordinates of the detected objects were sent to the user interface in a polar coordinate system. However, in that case, there was no method to estimate the distance to an object so only the angles and their probabilities were used to visualise the detections. In spite of that, in a more general scenario where a distance sensor was available, the orientation maps created by a robot would be more precise. Some of the examples are shown in Tables 6.4, 6.5, 6.6, which include images of the robot's and the objects' positions and the visualisation of the detections.

A similar validation procedure was performed to consider applying the vision system to a real device. To simulate a more realistic scenario where the classification pipeline may assist the robot in motion planning, the model was trained to recognise closed and open doors that are the objects that occur often in real indoor environments. The results are shown in Tables 6.7, 6.8, 6.9. Both experiments shown that the entire pipeline that has been discussed theoretically may be successfully applied in a real device

**Result of experiment 19.** *The recognition system was able to generate maps of estimated position of objects around the robot.*

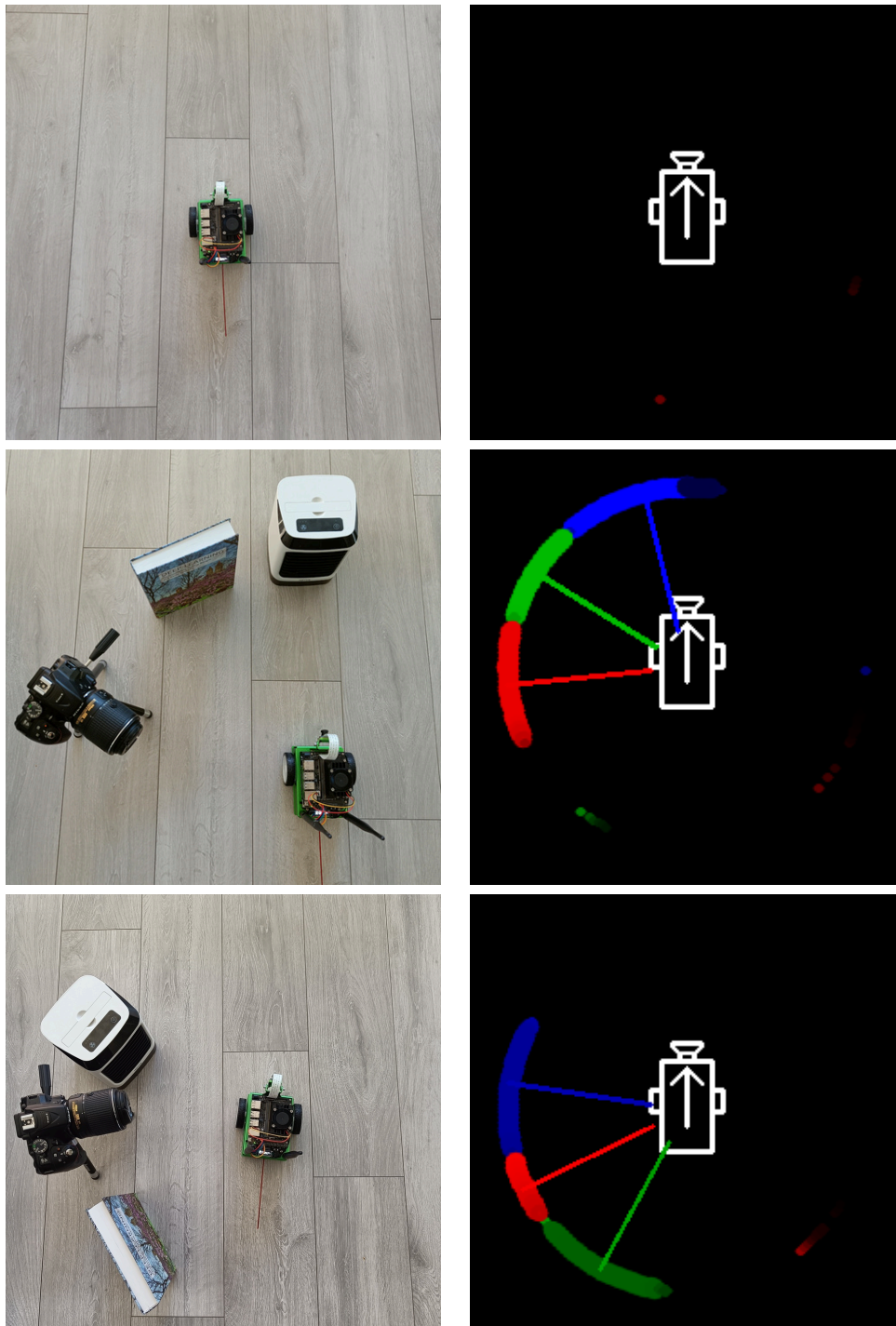


Table 6.4: Example recognition maps computed by mobile robot. Red dots denote camera, blue dots - AC, green dots - book. The bigger and brighter the dots the greater the certainty of occurrence.

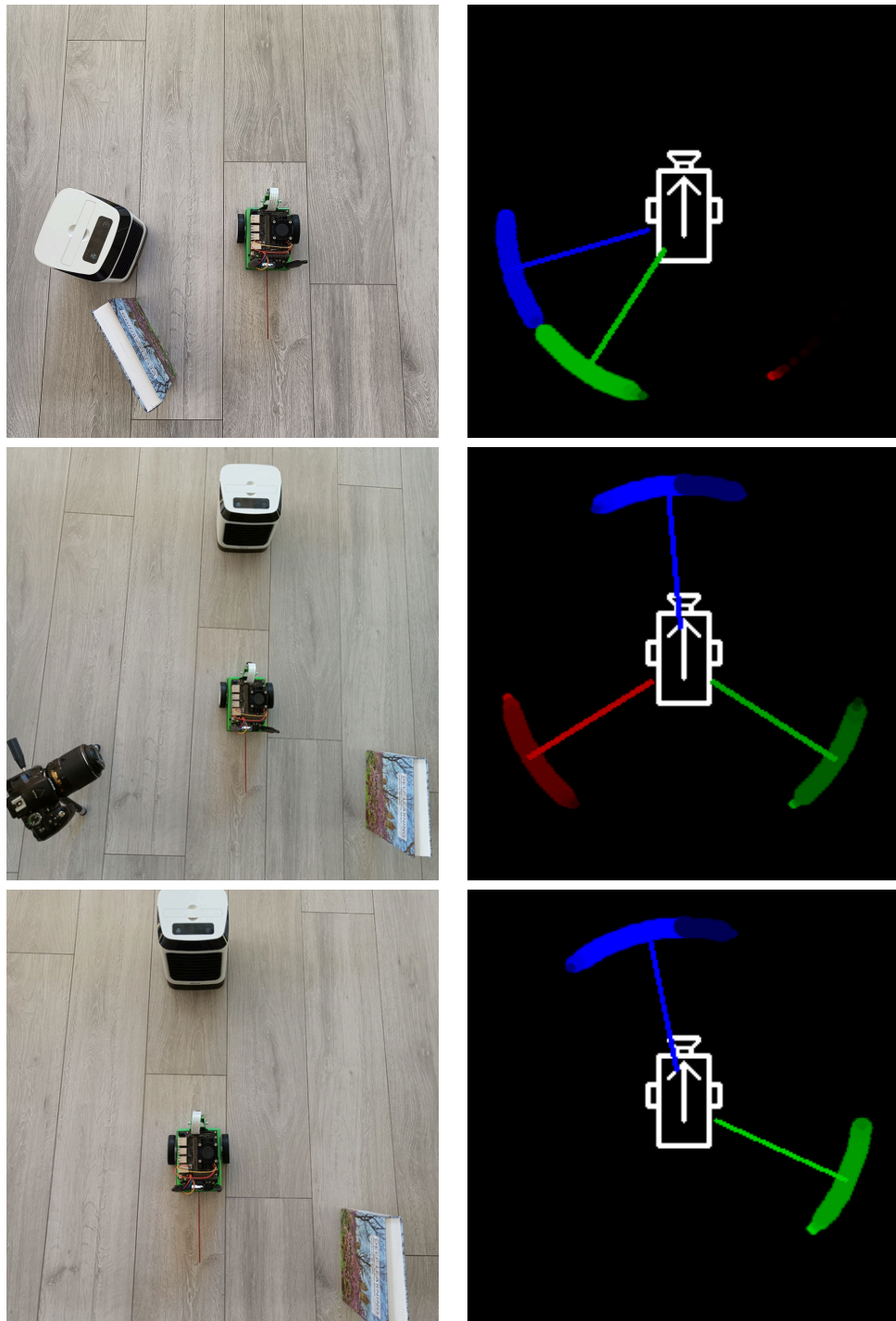


Table 6.5: Example recognition maps computed by mobile robot. Red dots denote camera, blue dots - AC, green dots - book. The bigger and brighter the dots the greater the certainty of occurrence.

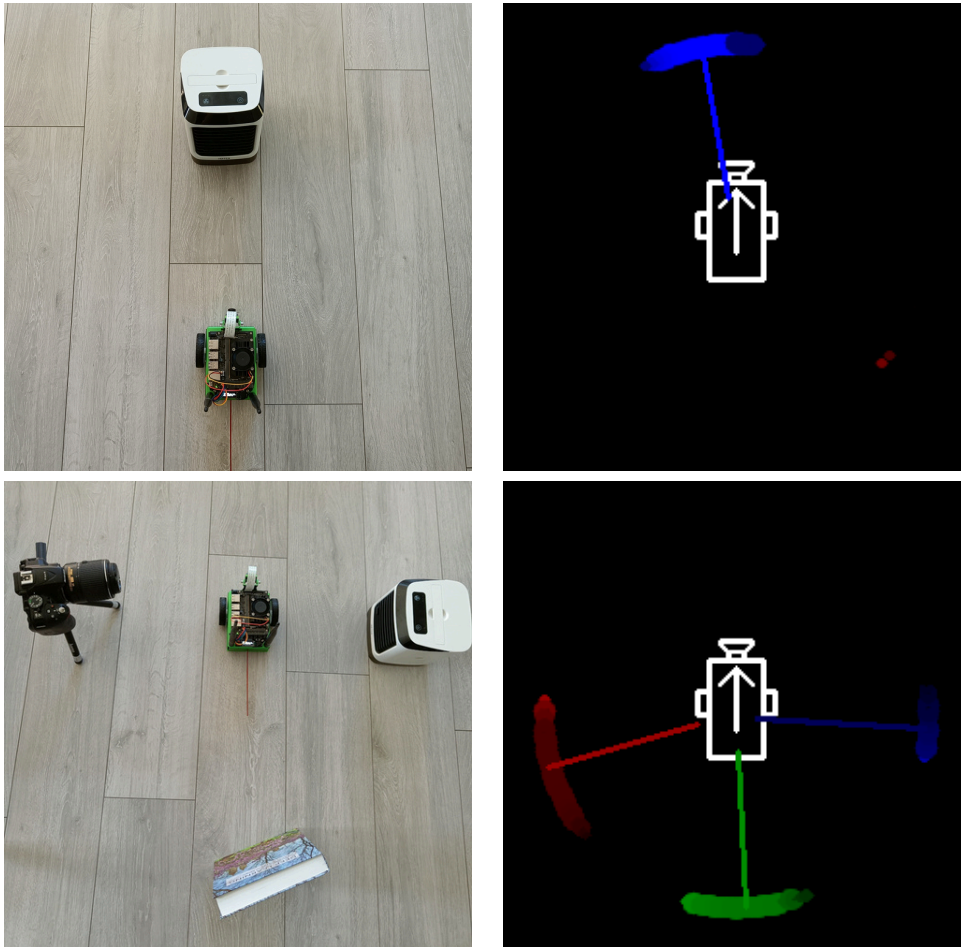


Table 6.6: Example recognition maps computed by mobile robot. Red dots denote camera, blue dots - AC, green dots - book. The bigger and brighter the dots the greater the certainty of occurrence.

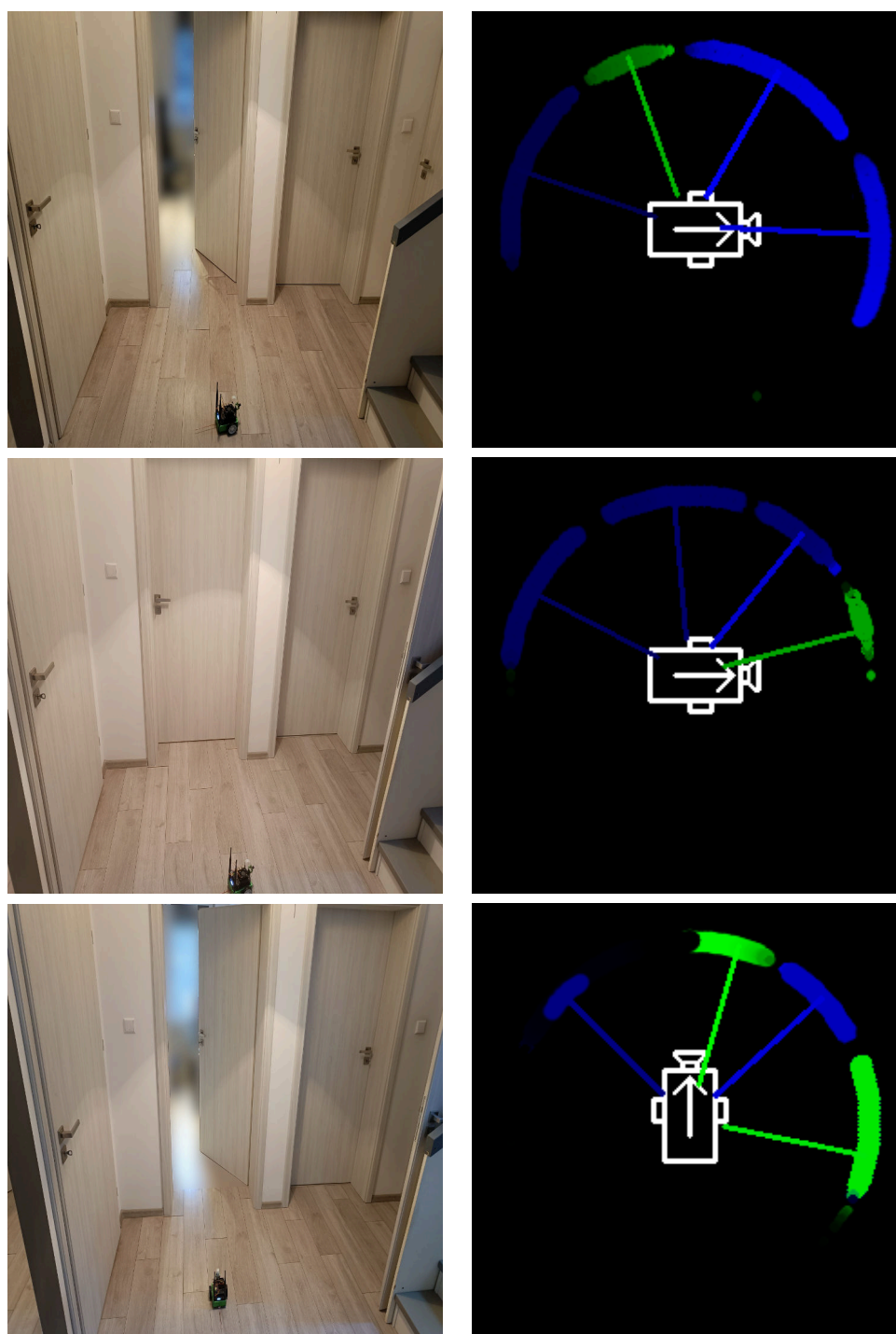


Table 6.7: Example recognition maps computed by mobile robot. Blue dots denote closed door, green dots - open door. The bigger and brighter the dots the greater the certainty of occurrence.

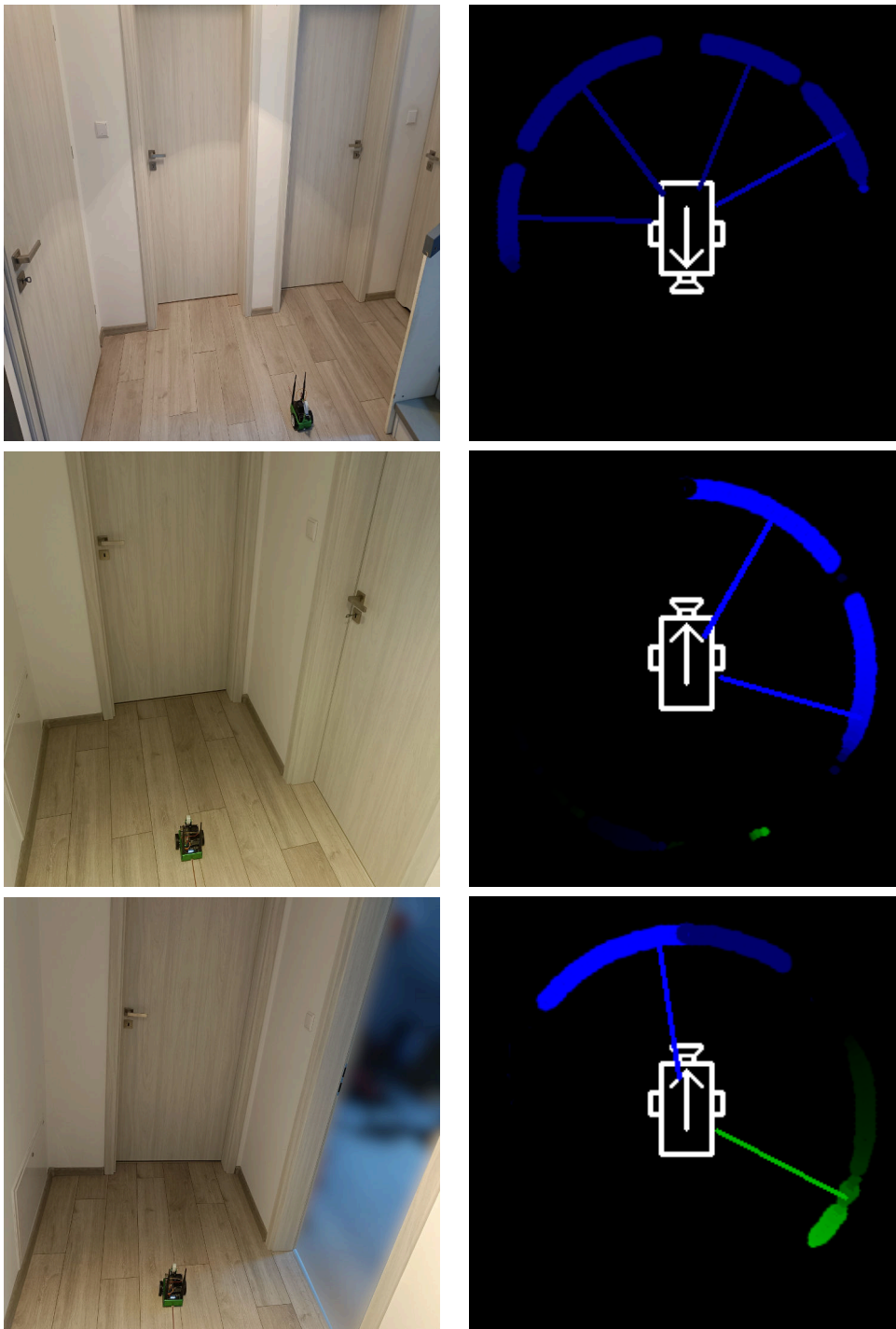


Table 6.8: Example recognition maps computed by mobile robot. Blue dots denote closed door, green dots - open door. The bigger and brighter the dots the greater the certainty of occurrence.



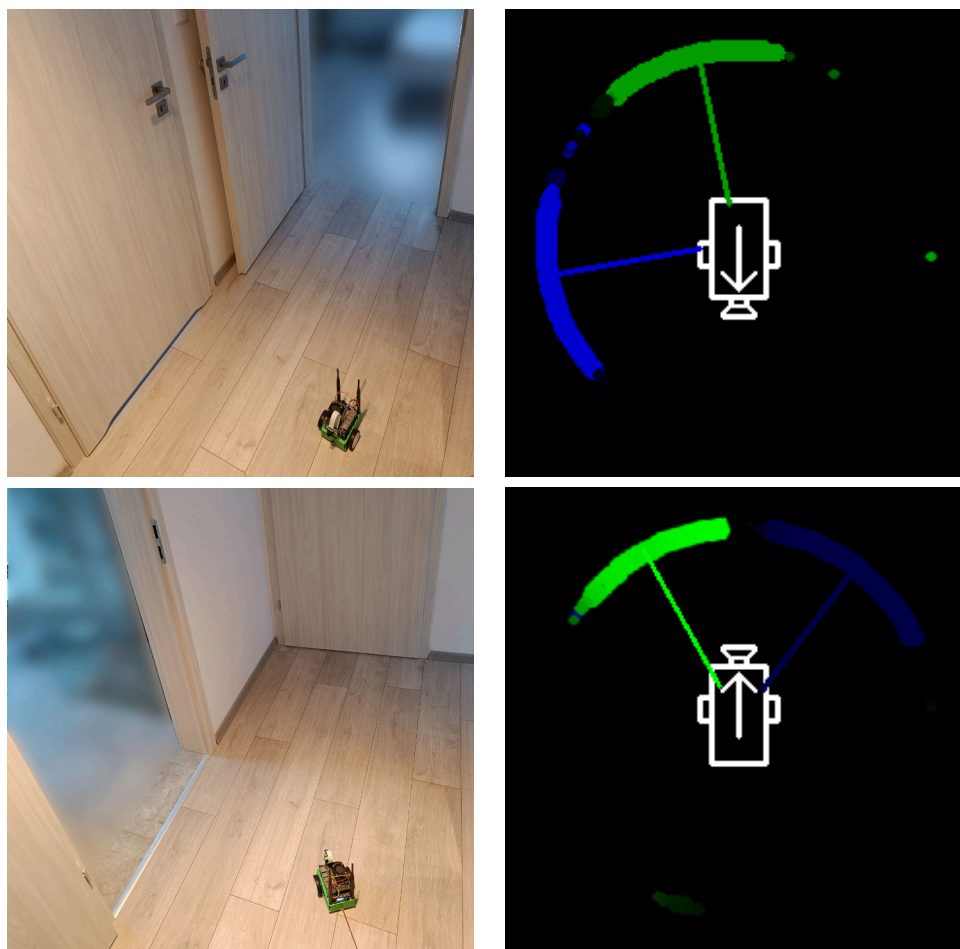


Table 6.9: Example recognition maps computed by mobile robot. Blue dots denote closed door, green dots - open door. The bigger and brighter the dots the greater the certainty of occurrence.

**Experiment 20 (Histogram based datasets comparison in the mobile robot’s recognition system).** *This experiment was designed to verify if the datasets similarity metrics could be applied for images gathered by the robot.*

The last experiment aimed validating if the proposed idea of using CLBP-RBM to measure the distance between datasets could be used in the robot’s application, and how that distance affected the overall CNN network performance in terms of transfer learning. Similar to the experiment described in Section 5.2, three datasets were created. The first was composed of images from regular house rooms (as for previous tests), the second was composed of frames gathered in a garden, and the third contained frames from a summer

house. Some example images from these sets are shown in Table 6.10. Intuitively, the first and the third are similar as they contain objects of similar semantic meaning (furniture, wall, doors, windows etc.) while the second differs significantly from as it is composed mostly of sky, trees, grass, plants, etc. Therefore, one can expect that the distance between the dataset from house to summer house is smaller than the distance to images from the garden. The results of the measurements based on the histograms comparisons are shown in Figure 6.11 and the Chi-Squared distances are summarised in Table 6.11. They verify the intuitive hypothesis of the similarity between datasets, one may conclude that the distance to images gathered from the garden is significantly higher than the other two which is observable in the histograms as well as in the Chi-Squared metric. The final test based on this was to validate how those RBMs would perform in the object recognition case, thus a CNN was trained with the use of CLBP-RBM preprocessing and three RBMs as a neural network in the preprocessing layer. Learning curves are shown in Figure 6.12. The conclusions are clear, the best accuracy was achieved by the RBM trained on the images from a house (same environment as for classification samples). However, the RBM that was close to it in terms of histograms distance worked well too while the RBMs trained on images from a garden achieved the worst performance. Therefore this showed experimentally the connection between the proposed metrics of similarity and the final accuracy of the CNN classification pipeline.

**Result of experiment 20.** *The datasets similarity metrics was applicable in mobile robot system and could be used to validate whether a given RBM could be transferred to a different problem or it had to be retrained.*

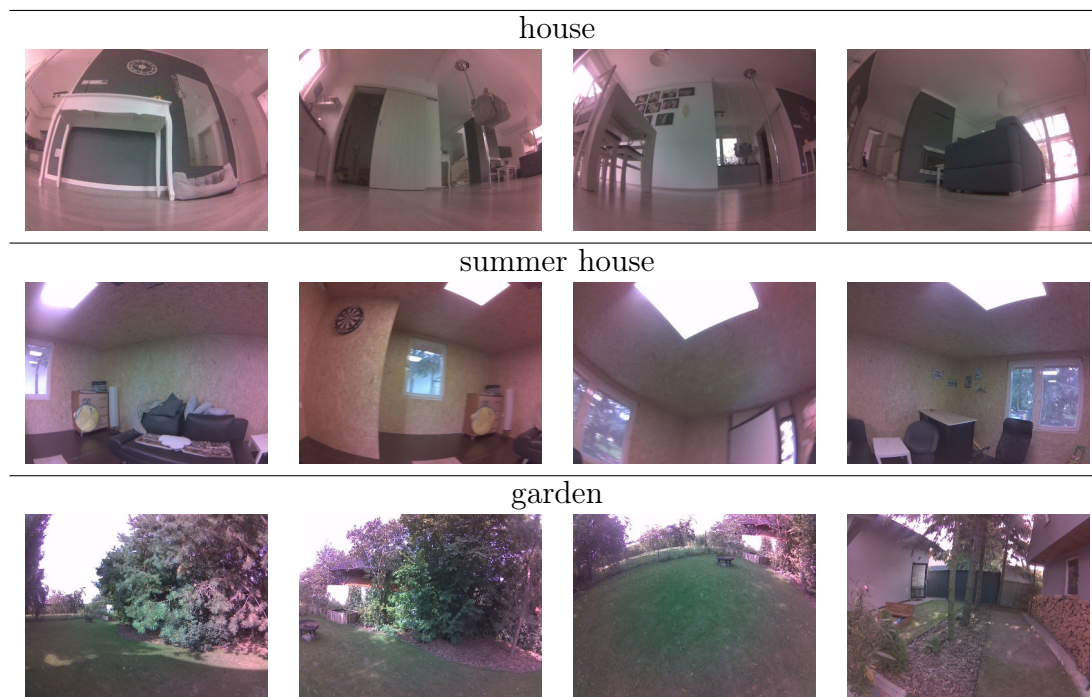


Table 6.10: Example images from different datasets used for experiments.

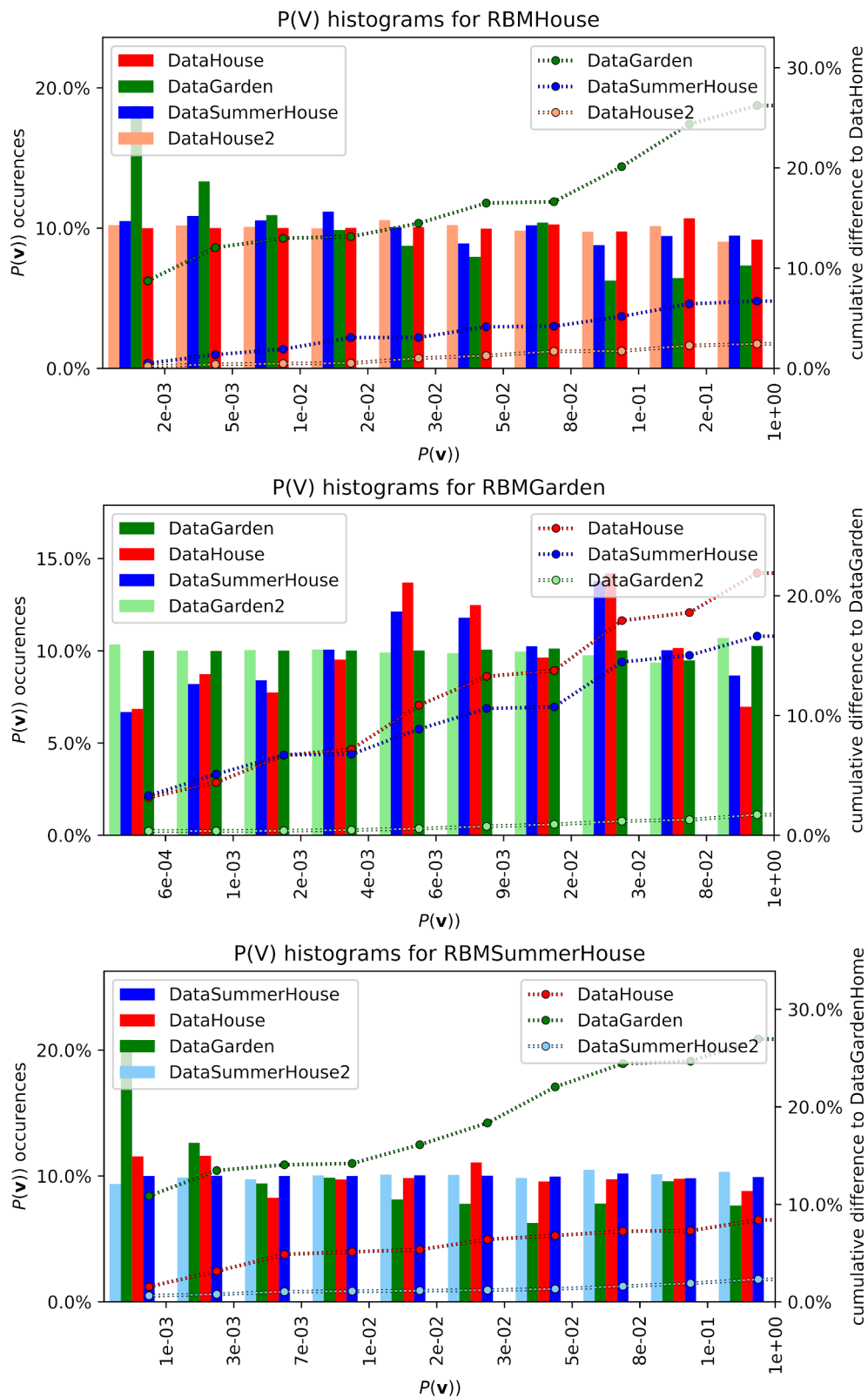


Figure 6.11: Histograms for  $P(\mathbf{v})$  obtained from different datasets.

	DataHouse	DataGarden	DataSummerHouse
RBMHouse	0.0010	0.1271	0.0063
RBMGarden	0.0649	0.0005	0.0410
RBMSummerHouse	0.0107	0.1583	0.0008

Table 6.11: Chi-Squared distances (defined in (4.18)) between the JetBot training datasets measured with the use of RBM's probabilities histograms, the diagonal results denote the distance between the reference set of images and another set from the same training dataset.

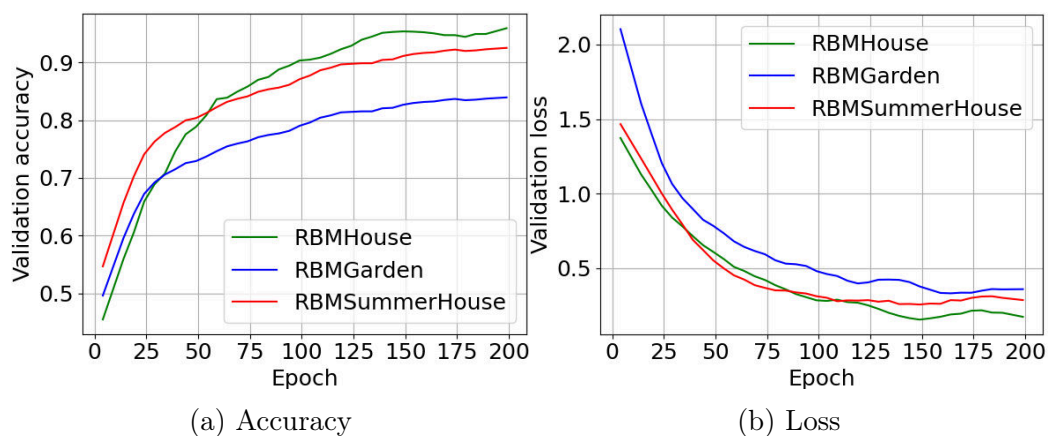


Figure 6.12: Training curves on the validation dataset for the mobile robot objects classification experiment with the use of different pretrained RBMs.



---

## CONCLUSIONS

---

This dissertation presents a novel method for using Restricted Boltzmann Machines (RBMs) to process local image binary descriptors, with the goal of improving the performance of image recognition tasks. The motivation for addressing that was mostly because of the gap in science in these descriptors' usability in modern classification methods. In short, local descriptors are robust unsupervised feature extractors but there were no efficient methods proposed in the literature that could utilise them for image recognition, the performance of known methods is rather not even comparable to current deep learning methods. The author argues that RBM can provide the high-level abstract meaning of a given image based on binary descriptors, which a deep neural network can further process for classification purposes. Furthermore, RBMs can be trained in an unsupervised manner, allowing the entire preprocessing to be also an unsupervised module in a classification pipeline. Intuitively, another disadvantage of descriptors is that they have a very general form of extracting the features, therefore they do not focus on the shapes that may be significant for a given set of samples. Processing them with an RBM focuses on dependencies between extracted features in a particular dataset, this model is capable of learning a distribution of training samples, hence it can amplify the significant features while suppressing the others. This type of data preprocessing is a unique method proposed by the author, also because of the lack of known RBM implementation that would meet this study requirements, it has been written from scratch.

There are many types of binary descriptors, however all of them share similar ideas and differ mostly in terms of how they code local image features. The author identified the colour Local Binary Pattern (CLBP) as the most efficient descriptor for the proposed preprocessing, as it provides information about the color of a given feature by comparing color pairs in the RGB color space. The enhancement is adding information about the colour of a given feature, which is achieved by comparisons of colour pairs from RGB colourspace. This is also a unique achievement of this study, and the

implementation of CLBP has been prepared by the author. Both the RBM and CLBP are written in Python and are fully parallel which allows running them in a highly efficient manner.

Processing binary patterns with RBM has been proposed mainly for image classification, however, the author of this study has shown other less obvious benefits of this method that stem from the fact RBM is not simply a model reinterpretable to a feed-forward network but has other capabilities as well.

The first is denoising of the input to a deep neural network, which is achievable because of the recurrent nature of RBM and the ability of data reconstruction. This is not a direct reconstruction of an image since it is firstly processed by a non-invertible descriptors transformation but it is the reconstruction of binary patterns which leads to better tolerance of a neural network to input distortion.

The second is a novel metrics of visual datasets comparisons, that can be used to measure a set of images (or two images). The author suggested that this may help in verifying whether an RBM trained on a set of images can be reused for other problems. Technically, this it is achievable based on another ability of an RBM, computing the probability of occurrence of a given input in the training dataset.

The author emphasised the two major concerns that may occur in modern image classification systems, the first is the lack of labelled data, and the second is the lack of processing units efficient enough to solve a given recognition problem. As suggested in this study both can be mitigated with the use of the proposed preprocessing method because of the unsupervised nature of the training and the simplicity. Finally, it implies its potential usability in robotics as this branch of technical science is especially vulnerable to these problems. Therefore, part of this dissertation is focused on transferring the results of the research on general datasets to a real device as an example of particular usage of this method. It is a mobile robot that has been running an efficient, fully parallel application written by the author. The application allows running the image processing pipeline together with a simple control loop and communication with a user interface running on a remote computer.

The classification methods proposed in this dissertation did not aim to outperform state-of-the-art deep neural classifiers in terms of overall accuracy, they focus on mitigating the previously mentioned practical problems. Therefore the experiments were mostly based on comparisons of how adding the CLBP-RBM preprocessing layer affects the generalisation ability of CNN. The analysis performed in Chapter 5 leads to the conclusion, that the preprocessing may improve the classification accuracy in some scenarios. Above all, they are most significant when the amount of labelled data is small or



when the CNN is relatively simple. Otherwise, for a large amount of well-structured data and very deep neural classifiers adding the CLBP-RBM layer does not affect the classification accuracy. This is because such cases do not suffer from the above-mentioned practical problems with insufficient number of training samples. However, the experiments showed that the preprocessing may result in a smaller CNN needed to solve a given problem with a given accuracy, hence there may be benefits in terms of network size optimisation. Other experiments also showed that the proposed novel binary descriptor - CLBP works best for the analysed problems and that its implementation is fast enough to claim that it does not affect the overall image processing time significantly. Likewise, the implementation of an RBM is also robust, especially when the number of its parameters is not large, so the CLBP-RBM layer may be potentially added to a classification pipeline without increasing the overall time of computing.

The mentioned ability to denoising the input data has been also experimentally confirmed. A number of distortions methods were tested, for most of them a network with CLBP-RBM layer achieved a better generalisation ability. The only exception was the compression noise where for very low quality of images using the network with CLBP-RBM preprocessing resulted in a higher decrease of accuracy.

The comparisons of image sets with the use of RBM were tested on three datasets and the hypothesis of their similarity has been verified. This means that the metrics for all the pairs of datasets were as expected, and the images being similar for humans were also similar in terms of this measurement.

The final experiments related to a mobile robot showed that moving the entire solution to an embedded device is possible and may provide a highly efficient real-time processing solution for visual perception. The robot was able to recognise different types of objects in less than 15 milliseconds. It has been also demonstrated that the same type of classifier operating on raw pixel data performs worse in terms of accuracy and with the use of a dedicated client-server application, it is relatively easy to gather images and train the neural networks without much human effort.

Processing binary descriptors with Restricted Boltzmann Machine has been proven effective in different scenarios and applicable to a variety of image processing problems. However, it is a very wide topic, thus there are many other areas where CLBP-RBM can be potentially used. The author limited the research to images classification, nevertheless, other tasks, such as object detection, key-points detection, pose estimation, segmentation, and many others that rely on convolutional feature extractors may benefit from the preprocessing as well since they share similar ideas in of terms how they process the input image. Likewise, a mobile robot is not the only field of

robotics and embedded systems that may utilise this type of preprocessing in terms of classification. Other cases, like manipulators, car cameras, drones, and humanoid robots are prone to the mentioned problems as well. Therefore, including an unsupervised module in their image processing pipeline that provides significant gains may result in improving their overall performance. Hence, the theoretical background given by the author may be further investigated in order to find other practical applications.

# A

---

## RBM AND CLBP IMPLEMENTATION DETAILS

---

The experimental research in this dissertation base primarily on the novel CLBP descriptor and Restricted Boltzmann Machine. In section 4.1, their implementations are described from a high-level point of view, however, some technical details are worth discussing.

The implementation of the CLBP descriptor is based on parallel computing with the use of CUDA [164] and NUMBA [156] as a Python library to translate a script into fast machine code. Thus the crucial part of the CLBP is a kernel that computes the output for entire input image, the code of such a kernel is presented in Listing A.1.

The RBM is written also in Python, however the parallel computing is performed with the use of TensorFlow [153]. The training procedure is independent on training the furthe CNN layers and reproducible with Algorithm 2. The output of such algorithm are parameters of an RBM (weights, biases of visual units and biases of hidden units), they may be stored in binary file (.npz) and reused later for inference with CNN. To provide the best possible performance a trained RBM may be embedded in CNN model as a regular TensorFlow layer, the code that reads RBM parameters from a file and implements RBM inference with optional descriptors denoising is presented in Listing A.2.

Listing A.1: Parallel computing of the CLBP descriptor.

```
# image - an input matrix of size (W x H x 3)
# output - an output matrix of size (W x H x 16)
@cuda.jit
def compute_CLBP(image, output):
    # for CLBP these parameters are fixed
    neighborhood = 1
    no_bits = 8

    y, x = cuda.grid(2)
    # skip computations for x,y being outside of the image
    if x >= image.shape[1] or y >= image.shape[0]:
        return
    # zero padding
    if x < neighborhood or y < neighborhood \
        or x + neighborhood >= image.shape[1] \
        or y + neighborhood >= image.shape[0]:
        output[y, x] = 0
        return

    # compute value of centre pixel in grayscale
    centre_pix = int((image[y, x, 0] + image[y, x, 1] + image[y, x, 2]) / 3)

    # LBP8 part
    a = 2 * np.pi * 1 / no_bits
    for b in range(no_bits):
        cord_X = int(r * round(np.cos((b - 5) * a)))
        cord_Y = int(-r * round(np.sin((b - 5) * a)))
        neighboring_pix = 0
        for channel in range(3):
            neighboring_pix += image[cord_Y, cord_X, channel]
        neighboring_pix = int(neighboring_pix / 3)
        output[y, x, b] = (centre_pix > neighboring_pix)

    # colours part
    T = 73
    for pair_nr in range(3):
        if pair_nr == 0:
            c0, c1 = 2, 1
        elif pair_nr == 1:
            c0, c1 = 0, 2
        else:
            c0, c1 = 1, 0
        output[y, x, 8] = image[y, x, c0] > image[y, x, c1]
        output[y, x, 9] = (image[y, x, c1] > (image[y, x, c0] + T)) \
            or ((image[y, x, c0] < image[y, x, c1] + T) \
                and (image[y, x, c0] > image[y, x, c1]))

    # intensity part
    output[y, x, 14] = (int(centre_pix) >> 7) & 1
    output[y, x, 15] = (int(centre_pix) >> 6) & 1
```

Listing A.2: Reusing RBM trained model as TensorFlow layer.

```

class RBMModule(tf.keras.layers.Layer):
    def __init__(self, beta=0.2):
        super(RBMModule, self).__init__(name="RBMEbedded")
        self.beta = beta
        # read data from .npz file
        data = np.load("RBM.npz")
        self.weights = tf.cast(tf.convert_to_tensor(data["weights"]), tf.float32)
        self.h_biases = tf.cast(tf.convert_to_tensor(data["h_biases"]), tf.float32)
        self.v_biases = tf.cast(tf.convert_to_tensor(data["v_biases"]), tf.float32)
        # reshape to match batch size dimension
        self.weights = tf.reshape(self.weights, (1, *self.weights.shape))
        self.v_biases = tf.reshape(self.v_biases, (1, *self.v_biases.shape))
        self.h_biases = tf.reshape(self.h_biases, (1, *self.h_biases.shape))

    def call(self, v, no_gibbs_steps=0):
        # cast from CLBP binary output
        v = tf.cast(v, tf.float32)

        # perform denoising when no_gibbs_steps > 0
        for step in range(no_gibbs_steps):
            ph = tf.matmul(v, self.weights)
            ph = tf.add(ph, self.h_biases)
            ph = tf.sigmoid(ph)
            h = ph > tf.random.uniform(tf.shape(ph), 0, 1)
            h = tf.cast(h, tf.float32)

            pv = tf.matmul(h, tf.transpose(self.weights, perm=[0, 2, 1]))
            pv = tf.add(pv, self.v_biases)
            pv = tf.sigmoid(pv)
            v = pv > tf.random.uniform(tf.shape(pv), 0, 1)
            v = tf.cast(v, tf.float32)

        # perform denoising when no_gibbs_steps > 0
        ph = tf.add(tf.matmul(v, self.weights), self.h_biases)
        ph = tf.tanh(tf.multiply(ph, self.beta))

        # reshape to a three-dimensional matrix w.r.t batch size
        # with the assumption that image is square,
        # otherwise the shape would have to be infered outside this function
        output_width_height = int(np.sqrt(pv.shape[1]))
        output_shape = (-1, output_width_height, output_width_height, pv.shape[2])
        pv = tf.reshape(pv, output_shape)
        return pv

```



# B

---

## TRAINING CONVOLUTIONAL NEURAL NETWORKS

---

Training a convolutional neural network was an inseparable part of the most of the experiments conducted in this thesis. This procedure was entirely implemented in TensorFlow with the help of OpenCV for some image processing operations. From a high-level point of view it was performed similarly to commonly known methods, thus it can be easily reproducible, however to present the training in more detail the simplified algorithm is schematically shown in Figure B.1.

The training procedure took advantage of batches to fit into memory and online data augmentation to prevent overfitting. After each training epoch the data was distorted to create a different input for the next learning iterations. The augmentation was performed with the use of Alumentations and OpenCV libraries [163, 191] and utilised the following methods: Gaussian noise, Gaussian blur, compression noise, rotation, horizontal flipping, random cropping, random channel amplification (RGB or HSV). As an optimisation algorithm RMSProp [198] was used.

An output from a training procedure was a *SavedModel* [153] which is the format recommended by TensorFlow. It can be also easily converted to a TensorRT [197] model which can provide a better performance on GPU, on NVIDIA Jetson Nano. This conversion reduced inference time by 50%.

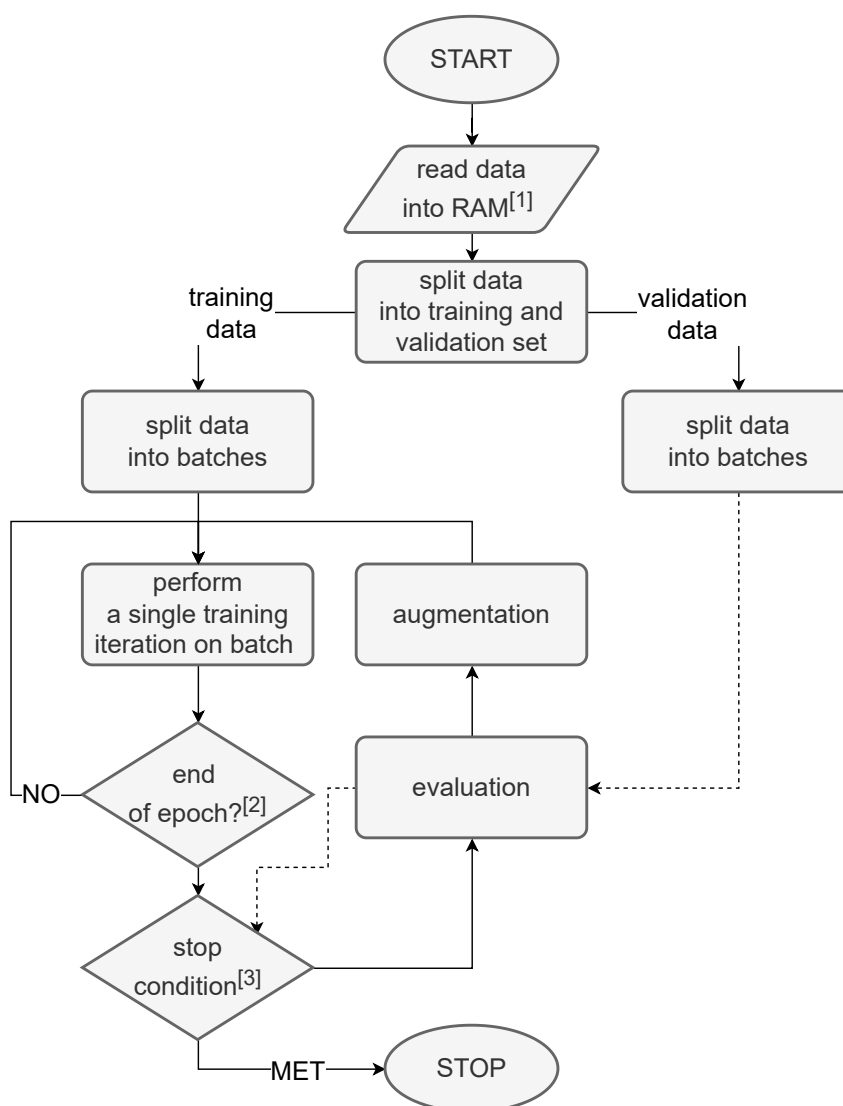


Figure B.1: Diagram of training convolutional neural networks,  
<sup>[1]</sup> - if dataset does not fit in RAM it is stored on disk and read into memory in batches,  
<sup>[2]</sup> - all batches from training dataset have been used,  
<sup>[3]</sup> - the number of epochs exceeded a predefined limit or validation loss does not decrease for a predefined number of epoch.



---

## ROBOT'S ANGLE DEVIATION MEASUREMENT WITH A DEDICATED VISION SYSTEM

---

In some of the experiments with the mobile robot, the orientation of the robot had to be measured. To make the method portable and allow gathering the statistics in a fast and simple way, a dedicated vision system that permits measuring an angular deviation from the desired position was prepared. The robot was equipped with its own marker - a thin red rod being an extension of its longer axis, and another marker - a yellow thin line was placed on the ground on the desired position of the robot. For such markers, it is possible to calculate the angle between the yellow line and the red marker ( $\theta$ ) by finding two points on each line. To demonstrate, the idea is presented schematically in Figure C.1.

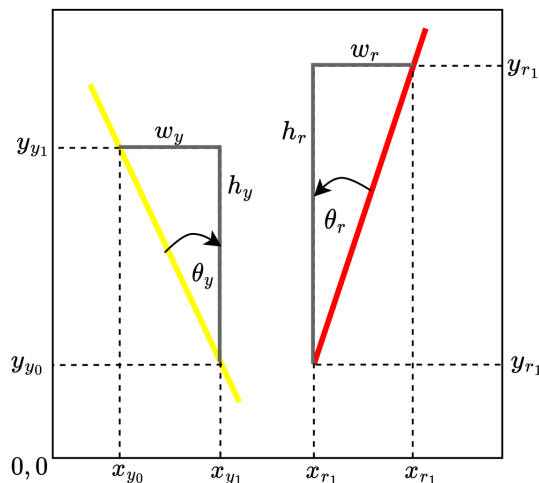


Figure C.1: Angle deviation estimation.

It is easy to observe that:

$$\begin{aligned}
 \theta &= \theta_r + \theta_y \\
 &= \operatorname{atan}\left(\frac{w_r}{h_r}\right) + \operatorname{atan}\left(\frac{w_y}{h_y}\right) \\
 &= \operatorname{atan}\left(\frac{x_{r1} - x_{r0}}{y_{r1} - y_{r0}}\right) + \operatorname{atan}\left(\frac{x_{y1} - x_{y0}}{y_{y1} - y_{y0}}\right)
 \end{aligned}$$

Algorithmically, a detection of  $[x_{r0}, x_{r1}, y_{r0}, y_{r1}, x_{y0}, x_{y1}, y_{y0}, y_{y1}]$  is relatively simple. The algorithm relies first on red/yellow colour separation, which is performed by transforming an input image to HSV colourspace and then using the following masks:

$$\begin{aligned}
 red &= [H[0 : 5] \ S[50 : 255] \ V[20 : 255]] \vee [H[175 : 180] \ S[50 : 255] \ V[20 : 255]] \\
 yellow &= [H[22 : 45] \ S[93 : 255] \ V[0 : 255]]
 \end{aligned}$$

Then on the red/yellow images, a Hough transformation [199] is performed to detect lines, that can be further used to infer needed points. In fact, to compute  $\theta$  any points from the detected lines can be chosen, but to make the method more precise, marginal points from each of the two lines should be taken. A simplified algorithm is presented in Figure C.2, and results from intermediate steps in Table C.1 and some example results in Table C.2.

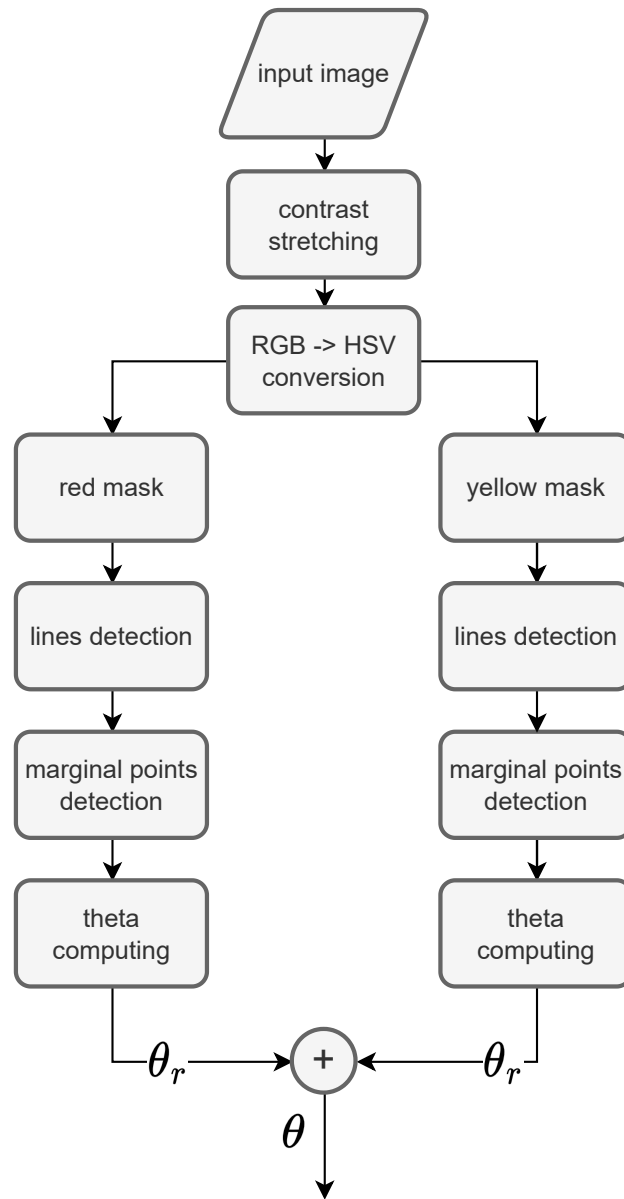


Figure C.2: Angle deviation algorithm.

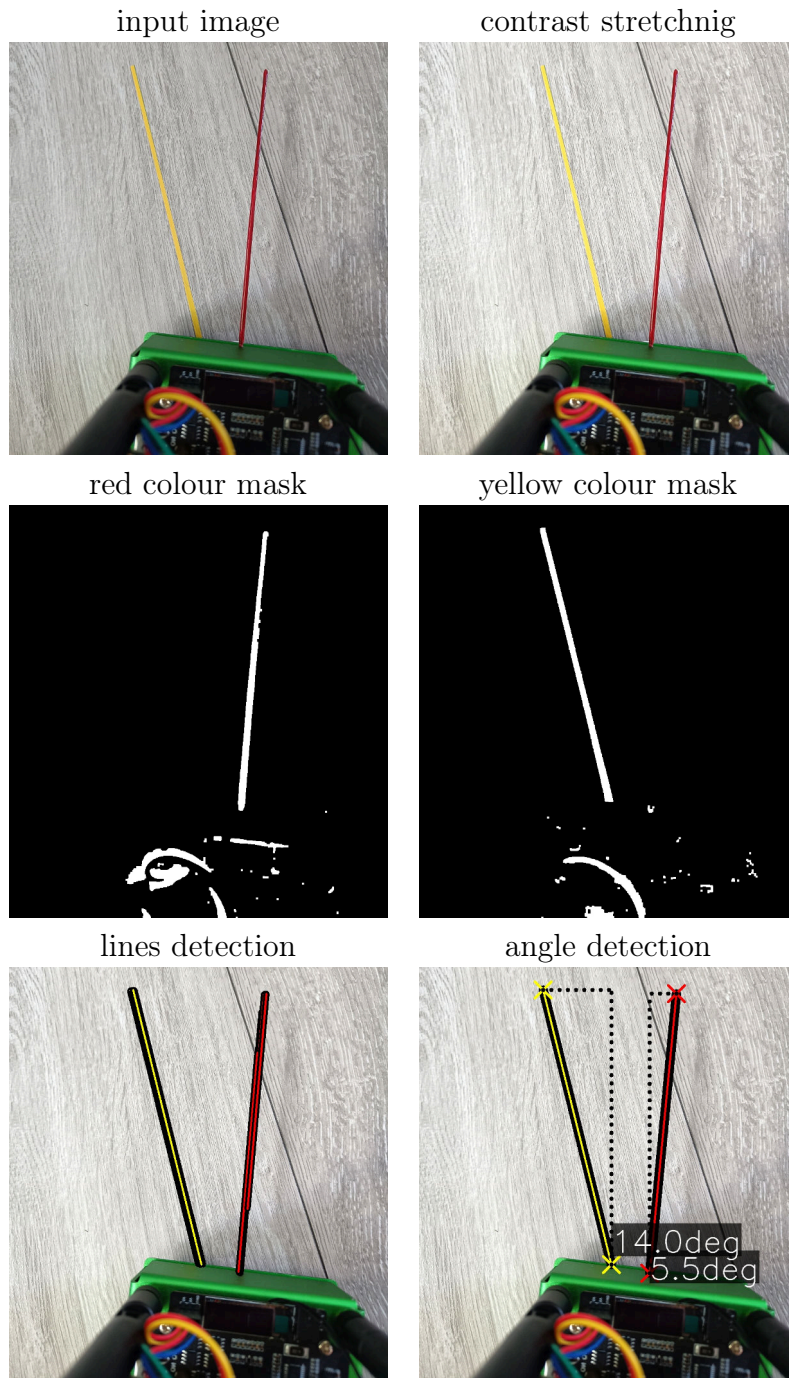


Table C.1: Intermediate results from angle deviation estimation.

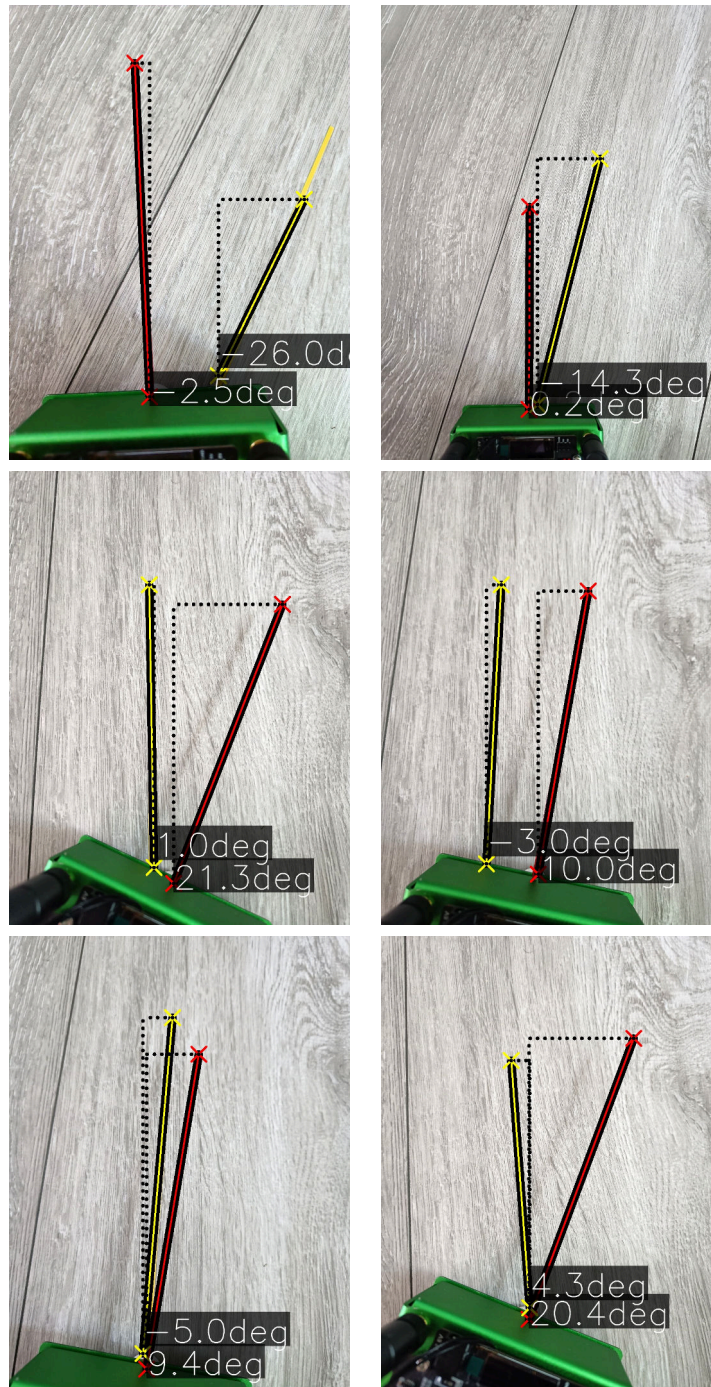


Table C.2: Angle deviation estimation results.



---

## Bibliography

---

- [1] R. Murphy, “An Introduction to Artificial Intelligence”, *Pearson Education*, 2012.
- [2] M. A. Srinivasa, D. Kuffner, J. C. Bo, J. M. Banks, “Robot Motion Planning and Control”, *Cambridge University Press*, 2010.
- [3] J. K. Aggarwal, Q. Cai, “Computer Vision: A Modern Approach”, *Pearson Education*, 2012.
- [4] R. Mur-Artal, J. D. Tardos, “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”, *IEEE Transactions on Robotics*, 2015.
- [5] D. Fox, W. Burgard, H. Durrant-Whyte, S. Thrun, “Markov localization for mobile robots in dynamic environments”, *Artificial Intelligence Journal*, 1997.
- [6] A. Y. Ng, M. I. Jordan, Y. Weiss, “On spectral clustering: Analysis and an algorithm”, *Advances in Neural Information Processing Systems*, 2002.
- [7] J. Z. Leibo, Y. W. Teh, D. M. Gavrilu, “Multi-view traffic sign recognition”, *Proceedings of the IEEE International Conference on Computer Vision*, 2015.
- [8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge”, arXiv:1409.0575, 2014.
- [9] W. McCulloch, W. Pitts, “A logical calculus of the ideas immanent in nervous activity”, *The bulletin of mathematical biophysics*, 1943
- [10] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain”, *Biology*, 1958.
- [11] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning”, *MIT Press*, 2016.

- [12] X. Glorot, A. Bordes, Y. Bengio, “Deep Sparse Rectifier Neural Networks”, *Journal of Machine Learning Research*, 2010.
- [13] C. E. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning”, *arXiv:1811.03378*, 2018.
- [14] Y. Bengio, P. Simard, P. Frasconi, “Learning long-term dependencies with gradient descent is difficult”, *IEEE Transactions on Neural Networks*, 1994.
- [15] P. Werbos, “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences”, *PhD thesis, Harvard University*, (1974).
- [16] D. Parker, “Learning Logic Report TR-47”, *MIT Press*, 1985.
- [17] Y. LeCun “Une procédure d’apprentissage pour Réseau à seuil asymétrique in *Cognitiva 85 a la Frontière de l’Intelligence Artificielle*”, *des Sciences de la Connaissance et des Neurosciences* [in French], 1985.
- [18] D. Rumelhart, G. Hinton, R. Williams, “Learning internal representation by error propagation”, *Learning Internal Representations by Error Propagation*, 1986.
- [19] A. Krizhevsky, I. Sutskever, G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, *Neural Information Processing Systems*, 2012.
- [20] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, L. Fei-Fei, “Imagenet: A large-scale hierarchical image database”. In *2009 IEEE conference on computer vision and pattern recognition*, (pp. 248–255), 2009.
- [21] Y. LeCun, Y. Bengio, G. Hinton, “Deep Learning”, *Nature*, 2015.
- [22] S. Bianchini, M. Müller, P. Pelletier, “Deep Learning in Science”, *arXiv:2009.01575*, 2020.
- [23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition”, *AT&T Bell Laboratories*, 1989.
- [24] B. Kwolek, “Face Detection Using Convolutional Neural Networks and Gabor Filters”, *Lecture Notes in Computer Science*, pp. 551-556, 2005.



- [25] K. Chellapilla, S. Puri, P. Simard, “High Performance Convolutional Neural Networks for Document Processing”, *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.
- [26] D. Cireşan, U. Meier, L. Gambardella, J. Schmidhuber, “Deep, big, simple neural nets for handwritten digit recognition”, *Neural computation*, pp. 3207-3220, 2010.
- [27] R. Girshick, J. Donahue, T. Darrell, J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [28] J. Long, E. Shelhamer, T. Darrell, “Fully convolutional networks for semantic segmentation”, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp 3431–3440, 2015.
- [29] A. Toshev and C. Szegedy. “Deeppose: Human pose estimation via deep neural networks”, *Computer Vision and Pattern Recognition (CVPR)*, pp 1653–1660, 2014.
- [30] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, “Large-scale video classification with convolutional neural networks”, *Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [31] N. Wang, D.Y. Yeung, “Learning a deep compact image representation for visual tracking”, *Advances in Neural Information Processing Systems*, pp 809–817, 2013.
- [32] C. Dong, C. Loy, K. He, X. Tang, “Learning a deep convolutional network for image super-resolution”, *In Computer Vision–ECCV 2014*, pp 184–199, 2014.
- [33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, “Going Deeper with Convolutions”, *arXiv:1409.4842*, 2014.
- [34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, “Rethinking the Inception Architecture for Computer Vision”, <https://arxiv.org/abs/1512.00567>, 2015.
- [35] K. Simonyan, A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, *International Conference on Learning Representations*, 2014.

- [36] H. Kaiming , X. Zhang, R. Shaoqing, S. Jian, Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [37] G. Huang, Z. Liu, L. van der Maaten, K. Weinberger, “Densely Connected Convolutional Networks”, *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [38] M. Tan, Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”, *arXiv:1905.11946*, 2019.
- [39] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, D. Keysers, J. Uszkoreit, M. Lucic, A. Dosovitskiy, “MLP-Mixer: An all-MLP Architecture for Vision”, *ArXiv, abs/2105.01601*, 2021.
- [40] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, Houlsby N., “An image is worth 16x16 words: Transformers for image recognition at scale”, *ICLR*, 2016.
- [41] N. J. Nillson, “Shakey the robot”, 1984.
- [42] H. Miyamoto, M. Kawato, T. Setoyama, R. Suzuki. “Feedback-error-learning neural network for trajectory control of a robotic manipulator”, *Neural Networks*, pp. 251-265, 1988.
- [43] C. Lin, C. Lee, “Neural-network-based fuzzy logic control and decision system”, *IEEE Transactions on Computers*, pp. 1320–1336, 1991.
- [44] W. Miller, P. Werbos, R. Sutton, “Neural networks for control”, *MIT Press*, 1995.
- [45] F. Lewis, S. Jagannathan, A. Yesildirak, “Neural network control of robot manipulators and non-linear systems”, *CRC Press*, 1998.
- [46] D. Pomerleau, “ALVINN, an autonomous land vehicle in a neural network”, *Advances in Neural Information Processing Systems*, 1989.
- [47] S. Joshi, S. Kumra, F. Sahin, “Robotic Grasping using Deep Reinforcement Learning”, *arXiv:2007.04499*, 2020.
- [48] S. Ainetter, F. Fraundorfer, “End-to-end Trainable Deep Neural Network for Robotic Grasp Detection and Semantic Segmentation from RGB”, <https://arxiv.org/abs/2107.5287>, 2021.

- [49] Y Jinglun, S. Yuancheng, L. Yifan, “The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning”, *Front Neurorobot*, 2020.
- [50] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, M. Ardani, “Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments”, *arXiv:2005.13857*, 2020.
- [51] K. Kolanowski, A. Świetlicka, R. Kapela, J. Pochmara, A. Rybarczyk, “Multisensor data fusion using Elman neural networks”, *Applied Mathematics and Computation*, pp. 236-244, 2018.
- [52] R. Kapela, A. Świetlicka, K. Kolanowski, J. Pochmara, A. Rybarczyk, “A set of dynamic artificial neural networks for robot sensor failure detection”, *2017 11th International Workshop on Robot Motion and Control (RoMoCo)* , pp. 199-204, 2017.
- [53] L. Canglong, Z. Bin, W. Chunyang, Z. Yongting, F. Shun, L. Haochen, “CNN-Based Vision Model for Obstacle Avoidance of Mobile Robot”, *MATEC Web of Conferences*, 2017.
- [54] P. Wenzel, T. Schön, L. Leal-Taixé, D. Cremers, “Vision-Based Mobile Robotics Obstacle Avoidance With Deep Reinforcement Learning”, *arXiv:2103.04727*. 2021.
- [55] T. Almeida, B Lourenço, V. Santosab, “Road Detection based on simultaneous Deep Learning Approaches”, *Robotics and Autonomous Systems*, 2020.
- [56] A. Narayan, E. Tuci, F. Labrosse, M. Alkilabi, “Road detection using convolutional neural networks”, *Proceedings of the 14th European Conference on Artificial Life ECAL 2017*, 2017.
- [57] Md. Z. Alom, M. T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, V. K. Asari, “The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches”, *arXiv:1803.01164*, 2018.
- [58] F. N. Iandola, S. Han., M. W. Moskewicz., K. Ashraf, W. J. Dally, K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size”, *arXiv:1602.07360*, 2016.
- [59] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, A. Hartwig, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, *arXiv:1704.04861*, 2017.

- [60] S. Bianco., R. Cadene, L. Celona, P. Napoletano, “Benchmark Analysis of Representative Deep Neural Network Architectures”, *arXiv:1810.00736*, 2018.
- [61] Md. Z. Alom, M. Tarek, C. Yakopcic, S. Westberg, “A State-of-the-Art Survey on Deep Learning Theory and Architectures”, *Electronics*, 2019.
- [62] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, “A Comprehensive Survey on Transfer Learning”, *arXiv:1911.02685*, 2019.
- [63] C. Cortes, V. Vapnik, “Support-Vector Networks”, *Machine Learning*, pp 273-297, 1995.
- [64] D.H. Hubel, N.T. Wiesel, “Receptive fields and functional architecture of monkey striate cortex”, *The Journal of Physiology*. pp 215-243, 1968.
- [65] I. Sobel, G. Feldman, “A 3x3 Isotropic Gradient Operator for Image Processing” *Pattern Classification and Scene Analysis*, pp. 271-272, 1973.
- [66] D. Scherer, A. Müller, S. Behnke, “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition”, *Artificial Neural Networks*, 2010.
- [67] A. K. Jain, R. C. Dubes, “Algorithms for clustering data,” *New Jersey: Prentice-Hall*, 1988.
- [68] K. Pearson, “On lines and planes of closest fit to systems of points in space.” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, pp. 559–572, 1901.
- [69] G. H. Golub, C. F. Van Loan. “Matrix Computation”. *Johns Hopkins Univ. Press, Baltimore*, 1996.
- [70] K. U. Gulden, G. Nese, “A Study on Multiple Linear Regression Analysis”, *Procedia - Social and Behavioral Sciences*, 2013, Vol. 106, pp. 234–240.
- [71] G.BakIr, B. Taskar, T. Hofmann, B. Scholkopf, A. Smola, S. Vishwanathan, “Predicting Structured Data”, *MIT Press*, 2007.
- [72] P. Baldi, K. Hornik “Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima”, *Neural Networks*, 1989, pp. 53–58.

- [73] D. Bank, N. Koenigstein, R. Giryes, “Autoencoders”, *arXiv:2003.05991*, 2020.
- [74] Y. Sun, H. Mao, Q. Guo, Z. Yi1, “Learning a good representation with unsymmetrical auto-encoder”, *Neural Computing and Applications*, 2016.
- [75] A. Majumdar, A. Tripath, “Asymmetric stacked autoencoder”, *2017 International Joint Conference on Neural Networks (IJCNN)*, 2020.
- [76] Y. Zhang, “A Better Autoencoder for Image: Convolutional Autoencoder”, *Computer Science*, 2018.
- [77] P. Kingma, M. Welling, “Auto-Encoding Variational Bayes”, *ICLR 2014 conference*, 2014.
- [78] K. He, X. Chen, S. Xie, Y. Li, P. Dollar, R. Girshick, “Masked Autoencoders Are Scalable Vision Learners”, 2021.
- [79] L. Girin, S. Leglaive, X. Bie, J. Diard, T. Hueber, X. Alameda-Pineda, “Dynamical Variational Autoencoders: A Comprehensive Review”, *Foundations and Trends in Machine Learning*, Vol. 15, No. 1-2, pp. 1-175, 2021.
- [80] A. Asperti, D. Evangelista, E. L. Piccolomini, “A survey on Variational Autoencoders from a GreenAI perspective,” *Computer Science*, Vol. 2, Issue 4, 2021.
- [81] J. J. Hopfield, J. J. Tank, “Computing with neural circuits: A model” *Science*, pp. 624–633, 1986.
- [82] D.O. Hebb, “The Organization of Behavior”, , 1945.
- [83] Y. Lecun, S. Chopra, R. Hadsell, “A Tutorial on Energy-Based Learning”, *Structured Data MIT Press*, 2006.
- [84] G. Pajares, M. Guijarro, A. Ribeiro, “A hopfield neural network for combining classifiers applied to textured images.” *Neural Networks*, 2010.
- [85] C. Hillar, R. Mehta, K. Koepsell. “A hopfield recurrent neural network trained on natural images performs state-of-the-art image compression.” *In 2014 IEEE International Conference on Image Processing (ICIP)*, pp 4092– 4096, 2014.

- [86] U. P. Wen, M.K. Lan, H. S. Shih. “A review of hopfield neural networks for solving mathematical programming problems”. *European Journal of Operational Research*, pp 675—687, 2009.
- [87] K. Smith, M. Palaniswami, M. Krishnamoorthy, “Neural techniques for combinatorial optimization with applications.” *IEEE Transactions on Neural Networks*, pp 1301–1318, 1998.
- [88] K. Tirdad, A. Sadeghian, “Hopfield neural networks as pseudo random number generators.”, *In 2010 Annual Meeting of the North American Fuzzy Information Processing Society*, pp 1—6, 2010.
- [89] S. M. Hameed, L. M. M. Ali, “Utilizing hopfield neural network for pseudo-random number generator”. *In 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, pp 1—5, 2018.
- [90] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, T. Adler, T. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, S. Hochreiter, “Hopfield Networks is All You Need”, *arXiv:2008.02217*, 2021.
- [91] D. Ackley, G. Hinton, T. Sejnowski, “A Learning Algorithm for Boltzmann Machines” *Cognitive Science*, pp. 147–169, 1985.
- [92] G. Hinton, “Boltzmann Machines” *Computer Science*, pp. 147–169, 2007.
- [93] P. Smolensky, “Information processing in dynamical systems: Foundations of harmony theory”, *MIT Press*, 1986.
- [94] K. Mikolajczyk, C. Schmid, “An affine invariant interest point detector”, *European conference on computer vision (ECCV)*, pp. 128–142, 2002.
- [95] A. Alahi, R. Ortiz, P. Vandergheynst, “Freak: Fast retina keypoint”, *Computer vision and pattern recognition (CVPR)*, pp=510–517, 2012.
- [96] T. Ojala, M. Pietikäinen, T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.
- [97] S. Leutenegger, M. Chli, R. Siegwart, “BRISK: Binary robust invariant scalable keypoints”, *International conference on computer vision (ICCV)*, pp. 2548–2555, 2011.

- [98] E. Rosten, T. Drummond, “Machine learning for high-speed corner detection”, *Computer Vision–ECCV 2006. Springer*, pp. 430–443, 2006.
- [99] M. Calonder, V. Lepetit, C. Strecha, P. Fua, “BRIEF: Binary robust independent elementary features”, *European conference on computer vision (ECCV)*, pp. 778–792, 2010.
- [100] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, “ORB: An efficient alternative to SIFT or SURF”, *International conference on computer vision (ICCV)*, pp. 2564–2571, 2011.
- [101] K. Grauman, T. Darrell, “The pyramid match kernel: Discriminative classification with sets of image features”, *Proc. ICCV*, 2005.
- [102] J. Zhang, M. Marszałek, S. Lazebnik, C. Schmid. “Local features and kernels for classification of texture and object categories: A comprehensive study.”, *Int’l J. Computer Vision*, pp. 213–238, 2007.
- [103] R. Fergus, P. Perona, and A. Zisserman. “Object class recognition by unsupervised scale-invariant learning”, *CVPR*, 2003
- [104] D. G. Lowe. “Distinctive image features from scale-invariant keypoints”, *Int’l J. Computer Vision*, pp. 91–110, 2004.
- [105] M. Vidal-Naquet, S. Ullman, “Object recognition with informative features and linear classification”. *Proc. ICCV*, 2003.
- [106] M. Brown, D. G. Lowe, “Automatic panoramic image stitching using invariant features”, *Int’l J. Computer Vision*, pp. 59–73, 2007.
- [107] S. Sobczak, R. Kapela, K. McGuinness, A. Swietlicka, D. Pazderski, N. O’Connor, “Restricted Boltzmann Machine as an Aggregation Technique for Binary Descriptors” *Visual Computer*, 2019.
- [108] R. Abdur, H. Najmul, W. Tanzillah, A. Shafiq, “Face Recognition using Local Binary Patterns (LBP).” *Global Journal of Computer Science and Technology Graphics & Vision*, 2013.
- [109] S. Moore, R. Bowden, “Local binary patterns for multi-view facial expression recognition”, *Computer Vision and Image Understanding*, pp. 541-558, 2011.
- [110] D. G. Lowe, “Object recognition from local scale-invariant features”, *International conference on computer vision (ICCV)*, Vol. 2, pp. 1150–1157, 1999.

- [111] H. Bay, T. Tuytelaars, L. Van Gool, “SURF: Speeded up robust features”, *European conference on computer vision (ECCV)*, pp. 404–417, 2006.
- [112] N. Dalal, B. Triggs, “Histograms of oriented gradients for human detection”, *Computer Vision and Pattern Recognition (CVPR)*, Vol. 1, pp. 886–893, 2005.
- [113] S. Beril, U. Cem, “Urban Area and Building Detection Using SIFT Keypoints and Graph Theory”, *IEEE Transactions on Geoscience and Remote Sensing.*, 2009.
- [114] S. Se, D. G. Lowe, J. Little, “Vision-based mobile robot localization and mapping using scale-invariant features”, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.
- [115] N. Dalal, B. Triggs, “Histograms of Oriented Gradients for Human Detection”, *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 886-893, 2005.
- [116] R. Kapela, P. Śniatała, A. Turkot, A. Rybarczyk, A. Pożarycki, P. Rydzewski, M. Wyczałek, A. Błoch, “Asphalt surfaced pavement cracks detection based on histograms of oriented gradients”, *2015 22nd International Conference Mixed Design of Integrated Circuits & Systems (MIXDES)*, pp 579–584. 2016
- [117] K. Horak, J. Klecka, O. Bostik, D. Davidek, “Classification of SURF image features by selected machine learning algorithms”, *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 636-641, 2017.
- [118] J. Sivic, A. Zisserman, “Video Google: A Text Retrieval Approach to Object Matching in Videos”, *International Conference on Computer Vision (ICCV)*, Vol. "2", pp. 1470–1477", 2003.
- [119] J. Philbin, O. Chum, M. Isard J. Sivic, A. Zisserman, “Object retrieval with large vocabularies and fast spatial matching”, *Computer Vision and Pattern Recognition (CVPR)*, 2007.
- [120] Hervé J, M. Douze, C. Schmid, P. Pérez, “Aggregating local descriptors into a compact image representation” *Computer Vision and Pattern Recognition (CVPR)*, pp. 3304–3311, 2010.



- [121] F. Perronnin, C. Dance, “Fisher kernels on visual vocabularies for image categorization”, *Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2007.
- [122] R. W. Hamming, “Error detecting and error correcting codes”, *Bell System Tech. J.*, pp 147-160, 1950.
- [123] C. Ken, S. Karen, V. Andrea, Z. Andrew, “Return of the devil in the details: delving deep into convolutional nets”, *In: Proceedings of BMVC*, 2014.
- [124] Y. Bengio, “Learning deep architectures for AI”, *Foundations and Trends in Machine Learning*, pp 1—127, 2009.
- [125] G. Hinton, R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, *Science 313(5786)*, pp. 504—507, 2006.
- [126] S. Nie, Z. Wang. Q. Ji, “A generative restricted Boltzmann machine based method for high-dimensional motion data modeling”, *Computer Vision and Image Understanding*, 2015.
- [127] R. Salakhutdinov, A. Mnih, G. Hinton, “Restricted boltzmann machines for collaborative filtering”, *Ghahramani, Z., ed.: ICML Volume 227 of ACM International Conference Proceeding Series.*, pp. 791—798, 2007.
- [128] K. Shimagaki, M. Weigt, “Selection of sequence motifs and generative Hopfield-Potts models for protein families,” *Physical Review*, Vol. 100, Issue 3, 2019.
- [129] A. Decelle, S. Hwang, J. Rocchi, D. Tantari, “Inverse problems for structured datasets using parallel TAP equations and restricted Boltzmann machines,” *Scientific Reports*, Vol. 11, 2021.
- [130] C. A. S. Assis, A. C. M. Pereira, E. G. Carrano, R. Ramos, W. Dias, “Restricted Boltzmann Machines for the Prediction of Trends in Financial Time Series,” *International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, 2018.
- [131] R. Kindermann, J. L. Snell, “Markov Random Fields and Their Applications”, *American Mathematical Society*, 1980.
- [132] D. Koller, N. Friedman. “Probabilistic Graphical Models: Principles and Techniques”, *MIT Press*, 2009.

- [133] A. Fisher, C. Igel, “An Introduction to Restricted Boltzmann Machines”, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 14–36, 2012.
- [134] O. Krause, A. Fisher, C. Igel, “Algorithms for estimating the partition function of restricted Boltzmann machines”, *Artificial Intelligence*, 2019.
- [135] F. Mazzanti, E. Romero, “Efficient Evaluation of the Partition Function of RBMs with Annealed Importance Sampling”, *arXiv:2007.11926*, 2020.
- [136] G. Hinton “A Practical Guide to Training Restricted Boltzmann Machines”, *Technical Report UTML TR 2010-003, University of Toronto*, 2010.
- [137] D. Ackley, G. Hinton, T. Sejnowski, “A learning algorithm for boltzmann machines”, *Cognitive Science*, pp. 147–169, 1985.
- [138] G. Hinton, M. A. Carreira-Perpiñán, “On Contrastive Divergence Learning”, *Computer Science*, 2005.
- [139] G. Hinton, “Training Products of Experts by Minimizing Contrastive Divergence”, *Neural Computation*, 2002.
- [140] G. Hinton, “Products of Experts”, *ICANN*, 1999.
- [141] L. Bottou, “On-line learning and stochastic approximations”, *On-line Learning in Neural Networks*, 1998.
- [142] G. Hinton, S. Osindero, Y. Teh, “A fast learning algorithm for deep belief nets”, *Neural Computation*, 2006.
- [143] G. Hinton, R. Salakhutdinov, “Deep Boltzmann Machines”, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, pp. 449–455, 2009.
- [144] G. Hinton, P. Dayan, B. Frey, R. Neal, "The wake-sleep algorithm for unsupervised neural networks", *Science*, pp. 1158-1161, 1995.
- [145] R. Salakhutdinov, H. Larochelle, “Efficient Learning of Deep Boltzmann Machines”, *Journal of Machine Learning Research - Proceedings Track*, pp. 693-700, 2010.
- [146] G. Taylor, G. Hinton, S. Roweis, “Modeling Human Motion Using Binary Latent Variables”, *Advances in neural information processing systems*, pp. 1345–1352, 2007.

- [147] A. Mohamed, G. Dahl, G. Hinton, “Deep belief networks for phone recognition”, *In Nips workshop on deep learning for speech recognition and related applications*, 2009.
- [148] A. Mohamed, G. Dahl, G. Hinton, “Acoustic modeling using deep belief networks”. *IEEE transactions on audio, speech, and language processing*, 2011.
- [149] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Journal of Machine Learning Research*, 2012.
- [150] B. Ghojogh, A. Ghodsi, F. Karray, M. Crowley, “Restricted Boltzmann Machine and Deep Belief Network: Tutorial and Survey”, *arXiv:2107.12521*, 2021.
- [151] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al., “Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, 2011.
- [152] C.R. Harris, K.J. Millman, S.J van der Walt, et al., “Array programming with NumPy”, *Nature*, 2020.
- [153] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, et al., “TensorFlow: Large-scale machine learning on heterogeneous systems”, <https://www.tensorflow.org>, 2015.
- [154] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, et al, “ PyTorch: An Imperative Style, High-Performance Deep Learning Library”, *Curran Associates, Inc.*, 2019.
- [155] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, “JAX: composable transformations of Python+NumPy programs”, <http://github.com/google/jax>, 2018.
- [156] S. K. Lam, A. Pitrou, S. Seibert, “Numba: A llvm-based python jit compiler.”, *In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015.
- [157] J. Uijlings, K. van de Sande, T. Gevers, A. Smeulders, “Selective Search for Object Recognition”, *International Journal of Computer Vision*, pp. 154–171, 2013.

- [158] H. Lee, R. Grosse, R. Ranganath, A. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations”, *roceedings of the 26th Annual International Conference on Machine Learning*, 2009.
- [159] D. Scott, “On optimal and data-based histograms”, *Biometrika*, pp. 605–610, 1979.
- [160] L. Fan, F. Zhang, H. Fan, C. Zhang, “Brief review of image denoising techniques”, *Visual Computing for Industry, Biomedicine, and Art volume*, 2019.
- [161] S. Dodge, L. Karam, “Understanding How Image Quality Affects Deep Neural Networks”, *arXiv:1604.04004*, 2016
- [162] G. Kylberg, I. M. Sintorn, “Evaluation of noise robustness for local binary pattern descriptors in texture classification”, *EURASIP Journal on Image and Video Processing*, 2013.
- [163] G. Bradski , “The OpenCV Library”, *Dr. Dobb’s Journal of Software Tools*, 2000
- [164] P. Vingelmann, F.H.P Fitzek, “CUDA, release: 10.2.89”, <https://developer.nvidia.com/cuda-toolkit>, 2020,
- [165] S. Sobczak, R. Kapela, “Restricted Boltzmann Machine as Image Pre-processing Method for Deep Neural Classifier” *2019 First International Conference on Societal Automation (SA)*, 2019.
- [166] S. Sobczak, R. Kapela, “Hybrid Restricted Boltzmann Machine– Convolutional Neural Network Model for Image Recognition” *IEEE Access*, 2022.
- [167] L. Fei-Fei, R. Fergus P. Perona, “Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories”, *CVPR 2004, Workshop on Generative-Model Based Vision*, 2004.
- [168] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, “Gradient-based learning applied to document recognition.”, *Proceedings of the IEEE*, 1998.
- [169] A. Krizhevsky, V. Nair, G. Hinton, “CIFAR-10 (Canadian Institute for Advanced Research)”, <http://www.cs.toronto.edu/~kriz/cifar.html>, 2009.

- [170] A. Coates, H. Lee, Y. Andrew, “An Analysis of Single Layer Networks in Unsupervised Feature Learning”, *AISTATS*, 2011.
- [171] A. Quattoni, A. Torralba, “Recognizing Indoor Scenes” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [172] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, A. Vedaldi, “Describing Textures in the Wild”, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [173] C. F. Özgenel, A. Gönenç Sorguç, “Performance Comparison of Pre-trained Convolutional Neural Networks on Crack Detection in Buildings”, *ISARC*, 2018
- [174] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, “Algorithms for Hyperparameter Optimization”, *Advances in Neural Information Processing Systems*, 2011.
- [175] M.A. Ranzato, C. Poultney, S. Chopra, Y. Lecun, “Efficient Learning of Sparse Representations with an Energy-Based Model”, *Neural Information Processing Systems Journal*, 2006.
- [176] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, M. Martina, “An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks”, *Future Internet*, Vol. 12, Issue 113, 2020.
- [177] X. Hu, L. Chu, J. Pei, L. Weiqing, B. Jiang, “Model complexity of deep learning: a survey”, *Knowledge and Information Systems*, Vol. 63, pp. 2585–2619, 2021.
- [178] A. Shaeiri, R. Nobahari, M. H. Rohban, "Towards Deep Learning Models Resistant to Large Perturbations", *arXiv, 2003.13370*, 2020.
- [179] A. Decelle, C. Furtlehner, “Restricted Boltzmann Machine, recent advances and mean-field theory”, <https://arxiv.org/abs/2011.11307>, 2020
- [180] O. Pele, M. Werman, “The Quadratic-Chi Histogram Distance Family”, *European Conference on Computer Vision*, 2010.
- [181] V. Srinivasan, Y. Han, S. Ong, “Image reconstruction by a Hofield neural network”, *Image and Vision Computing*, 1993.
- [182] M. Lin, Q. Chen, S. Yan, “Network In Network”, *Computer Science*, 2014.

- [183] P. Dutkiewicz, K. Kozłowski, W. Wróblewski, “Inspection robot SAFARI - construction and control”, *Bulletin of the Polish Academy of Sciences. Technical Sciences*, 2004.
- [184] L. Yang, B. Li, G. Yang, Y. Chang, Z. Liu, B. Jiang, J. Xiaol, “Deep Neural Network based Visual Inspection with 3D Metric Measurement of Concrete Defects using Wall-climbing Robot”, *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- [185] L. Yang, B. Li, G. Yang, Y. Chang, Z. Liu, B. Jiang, J. Xiaol “Automated wall-climbing robot for concrete construction inspection”, *Journal of Field Robotics*, 2022.
- [186] R. S. Lim, H. M. La, W. Sheng, “A Robotic Crack Inspection and Mapping System for Bridge Deck Maintenance”, *EEE Transactions on Automation Science and Engineering*, 2014.
- [187] J. Oh, G. Jang, S. Oh, J. Lee, B. Yi, Y. S. Moon, J. S. Lee, Y. Choi, “Bridge inspection robot system with machine vision”, *Automation in Construction*, 2009.
- [188] I. J. Goodfellow, J. Shlens, C. Szegedy, “Explaining and Harnessing Adversarial Examples”, *arXiv:1412.6572*, 2015.
- [189] A. K. Boyat, B. K. Joshi, “A review paper: noise models in digital image processing”, *Signal & Image Processing : An International Journal (SIPIJ)*, 2015.
- [190] M. M. P. Petrou, C. Petrou, “Image Processing: The Fundamentals”, *WILEY*, 2010.
- [191] <https://alumentations.ai/docs/>
- [192] <https://jetbot.org/master/>
- [193] <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>
- [194] <https://zeromq.org/get-started/>
- [195] <https://docs.python.org/3/library/tkinter.html>
- [196] R. Siegwart, I. R. Nourbakhsh, “Introduction to Autonomous Mobile Robots, Second Edition”, *MIT Press*, 2011.
- [197] <https://docs.nvidia.com/deeplearning/tensorrt/index.html>

- [198] T. Tieleman, G. Hinton, “Divide the Gradient by a Running Average of Its Recent Magnitude”, *Neural Networks for Machine Learning*, 2012.
- [199] R.O. Duda, P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures”, *Comm. ACM*, 1972.