

---

# Advances in Deep Learning for Vision, with Applications to Industrial Inspection

Classification, Segmentation and Morphological extensions

Doctoral Dissertation submitted to the  
Faculty of Informatics of the Università della Svizzera Italiana  
in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy

presented by  
Jonathan Masci

under the supervision of  
Prof. Jürgen Schmidhuber

March 2014



---

Dissertation Committee

<b>Prof. Michael Bronstein</b>	University of Lugano
<b>Prof. Illia Horenko</b>	University of Lugano
<b>Prof. Hugues Talbot</b>	Université Paris-Est, ESIEE
<b>Prof. Yann LeCun</b>	Courant Institute of Mathematical Sciences, New York University Director of AI Research, Facebook

Dissertation accepted on 27 March 2014

---

Research Advisor  
**Prof. Jürgen Schmidhuber**

---

PhD Program Director  
**The PhD program Director *pro tempore***

---

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

---

Jonathan Masci  
Lugano, 27 March 2014



*To my family*



# Abstract

Learning features for object detection and recognition with deep learning has received increasing attention in the past several years and recently attained widespread popularity.

In this PhD thesis we investigate its applications to the automatic surface inspection system of our industrial partner ArcelorMittal, for classification and segmentation problems. Currently employed algorithms, in fact, use fixed feature extractors which are hard to tune and require extensive prior-knowledge.

Our work, instead, focuses on learnable systems that can be used to improve recognition and detection without requiring hard to obtain task-specific domain knowledge.

For image classification we propose extensions to max-pooling convolutional networks, so that they can be applied to solve the general defect classification problem via a new pooling and feature encoding schemes.

State-of-the-art deep learning algorithms for object detection/segmentation have reached outstanding performance given high-quality annotated data. Unfortunately, they do not meet the required processing speeds of steel industry. We propose an architecture that does not suffer the same computational bottleneck (1500-fold speed-up) while retaining equal performance.

To further advance the field we study the learning of morphological operators, largely used in industry. Only few attempts have been proposed in the literature, but no approach has ever considered the problem in its generality because of its hard formulation. We tackle it from a different perspective and introduce a learnable framework which seamlessly integrates morphological operators; hence bringing these powerful tools to deep learning for the first time.

Re-engineering an industrial system requires time. In order to deliver an immediate return we investigate metric learning problems to boost performance of currently used features. Our multimodal similarity sensitive hashing model scales well to web-scale datasets and, thanks to the binary representation, requires little storage and involves a cheap distance computation. It outperforms previous state-of-the-art approaches without requiring additional resources.



# Acknowledgements

I would like to thank my advisor Prof. Jürgen Schmidhuber for his guidance and support during my work. I would also like to thank, with particular regards, Dr. Ueli Meier for having taught me how to be a good researcher and that I should never underestimate myself. Thanks also to Dr. Dan Ciresan without whom learning how to efficiently implement and use convnets would have took ages. I am grateful to all members of my institute, in particular to Varun Raj Kompella, Marijn Stollenga, Hung Ngo, Matt Luciw, Sohrob Kazerounian, Rupesh Srivastava, Alessandro Giusti and to all external collaborators I have had the honor and pleasure to work with. A special mention to Prof. Jesus Angulo and Dr. Gabriel Fricout for their valuable help and to Prof. Faustino Gomez for having taught me how improve my writing and presentation skills and for having spent his valuable time to review this manuscript. I would like to thank Prof. Michael Bronstein for the time and dedication demonstrated during our collaboration and finally all the reviewers who dedicated their valuable time to provide comments to improve this thesis.

Most importantly I would like to thank my family and my wife. Without their support I would have not been able to complete this PhD and it would have not had the same value.



# Contents

<b>Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>7</b>
2.1 Automatic Surface Inspection Systems . . . . .	8
2.2 Deep Learning for Vision . . . . .	12
2.2.1 Linear regression and multilayer perceptron . . . . .	14
2.2.2 Convolutional neural networks . . . . .	17
2.2.3 Learning strategies . . . . .	22
2.2.4 Preventing over-fitting . . . . .	25
2.3 Mathematical morphology . . . . .	31
2.3.1 Morphological operators . . . . .	32
2.3.2 Morphology in the steel industry . . . . .	39
2.4 Similarity based metric learning . . . . .	41
<b>3 Classifying defects with MPCNN</b>	<b>47</b>
3.1 Standard Features . . . . .	47
3.2 Dataset . . . . .	50
3.3 Experimental Setup . . . . .	52
3.4 Results . . . . .	54
3.4.1 Standard Features . . . . .	54
3.4.2 MPCNN . . . . .	54
3.4.3 Committee of classifiers . . . . .	56

<b>4</b>	<b>General steel defect recognition</b>	<b>57</b>
4.1	Background	58
4.1.1	Feature encoding algorithms	60
4.1.2	Feature pooling	61
4.2	Multi-Scale Pyramidal Pooling Network	62
4.2.1	Pyramidal pooling Layer	63
4.2.2	Multi-scale extraction	65
4.2.3	Feature encoding layer	66
4.3	Results	67
4.3.1	Conventional Benchmarks	68
4.3.2	Steel-Defects Industrial Benchmark	73
<b>5</b>	<b>Fast MPCNN image segmentation and detection</b>	<b>77</b>
5.1	Fast Image Scanning	80
5.2	Beyond patches: learning on full images	84
5.2.1	The MaxPoolingFragment (MPF) layer	85
5.2.2	Back-propagation through an MPF layer	85
<b>6</b>	<b>Application of MPCNN to steel segmentation and detection</b>	<b>89</b>
6.1	Results on single-channel images	89
6.1.1	Membrane Segmentation	90
6.1.2	Single Defect Detection	91
6.1.3	Multiple detections per image	94
6.2	Results on multi-variate images	96
<b>7</b>	<b>Learning Morphological Operators using Counter-Harmonic Mean</b>	<b>103</b>
7.1	Asymptotic morphology using Counter-Harmonic Mean	104
7.2	Method	105
7.3	Experiments	107
7.3.1	Learning dilation and erosion	108
7.3.2	Learning opening and closing	109
7.3.3	Learning top-hat transform	110
7.3.4	Learning denoising and image regularization	112
<b>8</b>	<b>Multimodal Similarity-Preserving Hashing</b>	<b>115</b>
8.1	Similarity-Preserving Hashing	117
8.1.1	Supervised single-modality similarity-preserving hashing	118
8.1.2	Supervised cross-modality similarity-preserving hashing	120
8.2	Multimodal NN hashing	120
8.2.1	Coupled siamese architecture	121



8.3 Experiments . . . . .	124
8.3.1 CIFAR10 . . . . .	125
8.3.2 NUS . . . . .	126
8.3.3 Wiki . . . . .	132
<b>9 Conclusions and Future Research</b>	<b>135</b>
9.1 Learning compact key-point image descriptors . . . . .	135
9.2 Morphological Operators for Structured Pooling . . . . .	137
<b>Bibliography</b>	<b>139</b>



# Figures

1.1	Sample images of two steel defects to illustrate the difficulties of the problem. On the left we have an extremely easy case, the defect is a vertical line, whereas on the right a challenging one, the defect is composed by the scattered spots. These images are taken from the current system and we clearly see that a big portion of the defect on the right is missing. . . . .	2
2.1	Illustrative example of a steel ASIS system processing pipeline. . .	8
2.2	Illustrative example of a steel ASIS acquisition system. Source: ArcelorMittal. The steel is inspected from both sides by linear cameras and carefully chosen light sources. . . . .	9
2.3	Illustrative example of several models which made the history of deep learning. We go from a single layer perceptron to the multilayer perceptron and reach the current state-of-the-art of deep networks which have many more layers of nonlinearity than conventional multilayer perceptrons. . . . .	15
2.4	A schematic representation of an MPCNN; the process flows left to right. Raw input pixel values are processed by a number of interleaved convolutional and max-pooling (MP) layers, which are trained to extract meaningful features. Several fully-connected layers (MLP) follow, which produce the final classification. . . . .	18
2.5	Examples of subtractive normalization on a conventional (left) and on a steel defect (right) image. . . . .	20
2.6	Examples of divisive normalization on a conventional (left) and on a steel defect (right) image. . . . .	20
2.7	Examples of subtractive divisive normalization on a conventional (left) and on a steel defect (right) image. . . . .	22

2.8	Examples of dilation and erosion with three different structuring elements. Leftmost column shows the original image, in this case we have a spot like defect and a vertically elongated one. Then from column 2–4 we have erosion (even rows) and dilation (odd rows) with respectively a vertical line of length 50, a disk of size 10 and a diamond of size 15. It is clear that with the dilation every bright structure that has at least a portion within the SE will be detected (white blobs); with the erosion the structure has to "fit" the SE in order to have a high response. . . . .	34
2.9	Examples of opening and closing with three different structuring elements as in Figure 2.8. Odd rows show the opening and even rows report the closing results. It is interesting to note the behavior on the opening with a vertical structuring element (second column): in the case of the spot defect it is completely removed from the image as the structure is too small to fit entirely the morphological kernel; for the vertically elongated defect instead we have a response on the actual defect and remove partially the noisy background. . . . .	36
2.10	Examples of opening by reconstruction using a disk structuring element of size 5 (central) and 50 (right). The original defected sample is shown on the left. Please note how the details are preserved even with such a strong image simplification. . . . .	38
2.11	The three classes of defects for which a morphological detector based on residual operators needs to be designed. . . . .	39
2.12	Detection results with residual operators on the images shown in Figure 2.11 . . . . .	40
3.1	(a): A schematic representation of how a LBP descriptor is computed; (c) exemplar LBP image when applied to the steel defect image (b). . . . .	48
3.2	Visualization of HOG descriptors on industrial steel defect images. The detector is able to capture interesting points in the image and describes them with their gradient orientation. In cases where the background is highly textured, however, such an approach tends to fail. . . . .	49
3.3	Two instances of the same defect class as labeled in our dataset. .	50

3.4	Each point denotes the width and height of an image from the training set, histograms of the width and height distribution are also shown. It can be seen that most of the images are smaller than 150px in width and 200px in height. This distribution is used to empirically select the nominal size of the input images for the MPCNN. . . . .	51
3.5	A sample from each of the seven defects in the dataset. First row: superior part of the strip; second row: inferior part of the strip. There is no correspondence between the two images for a given instance of the defect. . . . .	52
3.6	Confusion matrices for the best classifiers. Left: MPCNN, middle: PHOG, right: PHOG + MONO-LBP committee. Only on defect number two the classical features obtained a better result than that of the MPCNN. Also note the non marginal improvement of a committee w.r.t. the single best classifier. . . . .	56
4.1	Schematic representation of a MSPyrPool Network where histogram-like representations are extracted at two levels and at two scales. The first scale represents the output of the convolutional layer whereas the second scale is given by the output of a pooling (downsampling) layer. The resulting features are concatenated and used as input for the classification layer. . . . .	62
4.2	Pyramidal Pooling Layer. Features at each level $l$ of the pyramid are pooled along $l^2$ equally sized quadrants and the histogram-like representations are concatenated to form a feature vector. . .	64
4.3	The MLPdict layer used for feature encoding. Image responses of a network layer are reshaped to produce $K$ dimensional feature vectors, where $K$ represents the number of images in the layer. Each pixel descriptor is mapped into another representation of size $K'$ for which only the maxima value per row is preserved. . .	67
4.4	Some exemplar images from the MNIST digit recognition dataset.	69
4.5	Some exemplar images from the CURET texture recognition dataset. . . . .	70
4.6	Some exemplar images from the Caltech101 texture recognition dataset. . . . .	72
4.7	Subset of images from the steel-defects benchmark showing the great difference in size among various samples. In this setting is not possible to resize the images to the same size, hence a MPCNN is not applicable to solve this task. . . . .	74

4.8	Two misclassified images for which the network inverted the corresponding classes (off diagonal elements in the confusion matrix). This example clearly shows the extreme difficulty of this task. . . . .	75
5.1	Illustrative example of how much redundant computation there is when testing a MPCNN on a patch-by-patch basis. Similarly such situation is encountered also during training. Yellow and blue patches are input to the same MPCNN which computes twice the same intermediate results indicated by the diagonal stripes. When a max-pooling operation is applied, given its partial result it is possible to classify only the pixel indicated in the corresponding output; e.g. A for yellow and B for blue. . . . .	79
5.2	MPF layer for the $2 \times 2$ pooling case. <i>Top</i> : Forward pass, <i>Fragments</i> (0,0) and (1,1) share the same maximal element; <i>Bottom</i> Back-propagation pass where partial derivatives are pushed back to the previous layer in the hierarchy; the partial results of each <i>Fragment</i> are summed together. . . . .	86
5.3	Illustrative example of how subsequent convolutional layers of a MPF layer work. The same operation is applied to each output fragment; the gradient is the sum of the gradients of the fragments. . . . .	87
6.1	A slice of the test set segmented using SEGNN trained on full images. The image on the right shows the probability of each pixel to be assigned to the background (white) or to the membrane (black). Please refer to text for the network architecture details. . . . .	91
6.2	A typical steel defect example for the marker detection problem. The marker, whose location is shown in the target image, needs to be detected and a flag must be raised when the input image contains it. We can see that segmentation is almost perfect, illustrating the power of the proposed approach for industrial applications. . . . .	93
6.3	Precision-Recall curves for the three methods on the bobine dataset. We clearly see that SEGNN has a much better precision for any recall value. Of particular interest is the good performance when recall is high as it means that less false positive alarms are raised. . . . .	95

6.4	A typical steel defect example from the coil dataset. This is a more challenging problem than the one of Sec. 6.1.2. Given the images in the first column we are asked to produce the ROI as indicated in the second column, the ground-truth data. SEGNN results, for these test images, are reported in the third column. The top-most result is almost perfect. For the second image from the top, the long vertical stripe of SEGNN, even if larger than desired, after a visual inspection makes lot of sense as it closely capture the defected region. . . . .	96
6.5	Signatures used for the multi-variate dataset generation. Note that target C and D are extremely difficult to discriminate, they have no relevant peaks or differences but belong indeed to two different defect classes. . . . .	97
6.6	Example images from the multi-variate synthetic dataset of steel defects. Note that the data has 23 channels, but only channels 1, 10 and 23 are plotted in RGB for illustrative purpose only. Target images, second and fourth column from the left, show the defect and its corresponding color coded class label. Each defect, such as the red one for example, can come in different shapes so that learning the shape of the object does not suffice to obtain good performance. The first and third columns show the images which compose the synthetic dataset. . . . .	98
6.7	Precision-Recall curves for the three methods on the multi-variate dataset for the detection task where the SEGNN detector outperforms by a large margin its best competitor. Please note that SEGNN does not require any knowledge on the data whereas the Target Detector needs the signature of the defects. We implicitly solve the related inverse problem of finding such signatures through learning the segmentation map, in our opinion a remarkable achievement of our model. . . . .	99
6.8	Precision-Recall curves for the three methods on the multi-variate dataset where class 4 is labeled as background. When the 23-channel raw signal is used SEGNN achieves almost perfect precision at very high recall, contrary to other approaches which sharply decay. . . . .	101

7.1	Illustrative example of a PConv layer. The network here represented can be used for learning dilation and erosion. In case of multiple convolutional kernels (i.e. multiple structuring elements in this setting) there will be a $P$ for each one of them. This way it is possible to learn a richer set of operators. . . . .	107
7.2	Examples of learning a dilation with three different structuring elements. The target and net output are slightly smaller than the original image due to valid convolution. The obtained kernel $w(x)$ for each case is also depicted. . . . .	109
7.3	Examples of learning an erosion with three different structuring elements along with the learned kernel $w(x)$ . . . . .	110
7.4	<i>Top</i> : an example of learning a closing operator where a line of length 10 and orientation $45^\circ$ is used. <i>Bottom</i> : opening with a square structuring element of size 5. The network closely matches the output and almost perfectly learns the structuring elements in both PConv layers. . . . .	111
7.5	Learning a top-hat transform. The defected input image has bright spots to be detected. The network performs almost perfectly on this challenging task. . . . .	112
7.6	Learning two top-hat transforms. On the left, bright spots need to be detected. On the right, a dark vertical line. The network performs almost perfectly in this challenging task. . . . .	112
7.7	<i>Top</i> : Binomial noise removal task. The learned nonlinear operator performs better than the hand-crafted one by a relative margin of $\approx 7.7\%$ . Learning uses noisy and original images—there is no prior on the task. <i>Bottom</i> : Salt’n’pepper noise removal task. Even here, the learned operator performs better than the corresponding morphological one. . . . .	113
7.8	Total Variation (TV) task. The network has to learn to approximate the TV output (target) by means of averaging two filtering pipelines. . . . .	114
8.1	Schematic representation of the coupled siamese network. There are two networks, one for each of the modalities, coupled by the loss function $L_{xy}$ . . . . .	122



8.2	Unimodal retrieval on CIFAR dataset. Shown are top 10 matches to three different queries (marked in red) using different hashing method with codes of length 48. All NN methods are used in L2 configuration. CM-SSH, CM-NN and MM-NN were trained on multiple modalities, and used in this experiment for single modality retrieval. . . . .	126
8.3	Example of GIST-HOG matching on CIFAR dataset. Shown are the original descriptors and their 48-bit MM-NN L2 hash codes. Red shows the bits that are different w.r.t. the query. . . . .	128
8.4	Precision-Recall curves for the cross-modal retrieval experiments on CIFAR10 (solid: HOG-GIST, dashed: GIST-HOG) and NUS (solid: Tag-Bof, dashed: Bof-Tag). . . . .	131
8.5	Example of text-based image retrieval on NUS dataset using multimodal hashing. Shown are top five image matches produced by CM-SSH (odd rows) and MM-NN (even rows) in response to three different textual queries. . . . .	132
8.6	Example of image annotation on the NUS dataset using multimodal hashing. Shown are Tags returned for the image query on the left. Groundtruth tags are shown in green. . . . .	133
8.7	Cross-modal (Bof-Tags) retrieval on the NUS dataset. Shown are top five matches different image queries (marked in red), ranked according to Tags similarity using 64-bit MM-NN hash. . . . .	133
8.8	Cross-modal (Image-Text) retrieval on the Wiki dataset. Shown are top five matches different image queries (marked in red), ranked according to text similarity using 32-bit MM-NN hash. . .	134
9.1	Typical pipeline of feature descriptor construction, consisting of: affine-invariant region detection and canonization, ensuring approximate invariance to view point transformations; linear filtering part (e.g. gradient computation in SIFT), ensuring illumination invariance; non-linear part (e.g. local directions histogram in SIFT). Hua <i>et al.</i> focused on tuning the parameters of the linear and non-linear parts of SIFT. Strecha <i>et al.</i> added another binarization stage. . . . .	136



# Tables

3.1	Detailed networks structure. The time per sample refers to the time required for a trained network to produce the class prediction.	54
3.2	Classification results for the several methods on the steel classification dataset. Classical features show results as SVM/MLP. For MPCNN the best run is presented along with, in parenthesis, the mean performance and standard deviation among 5 different runs. RC indicates that the first convolutional layer performs a random projection, not trained.	55
4.1	Classification results for the MNIST benchmark. The MSPyrPool network is compared with other CNN-based approaches which do not use any input preprocessing.	69
4.2	Classification results for the CURET benchmark. A conventional MPCNN is compared with our MSPyrPool network. We also show the relative improvement of a MLPDict encoding stage.	71
4.3	Classification results for the Caltech101 benchmark. A MSPyrPool net without an encoding layer (MSPyrPool1), with a linear (MSPyrPool2) and a nonlinear encoding layer (MSPyrPool3) are listed.	72
4.4	Classification results for the Steel-defect benchmark where MPCNN fail. Various classifiers, trained on conventional features, are compared with the MSPyrPool network which outperforms, by large margin, any classifier based on engineered features.	73
5.1	Theoretically required FLOPS for convolutional layers when segmenting a $512 \times 512$ image using patch-based ( $\text{FLOPS}_{\text{patch}}^l$ ) and image-based ( $\text{FLOPS}_{\text{image}}^l$ ) approaches.	83
5.2	Speed for segmenting a $512 \times 512$ image using the large net described in <a href="#">Ciresan et al. [2012a]</a> .	84

6.1	Comparison of training times for the Membrane dataset. The overhead for generating the transformed samples is also included in the overall computation. The relative speed-up of SEGNN is shown in parenthesis. . . . .	91
6.2	Detection error results of our efficient learning framework for MPCNN. Test evaluation times for a given image are also reported along with the patch-based evaluation with the same implementation (e.g. Matlab). . . . .	93
6.3	Average Precision results for several models on the full steel coil detection problem. All methods work on the same support of $32 \times 32$ pixels. . . . .	95
6.4	Average Precision results for several models on the multi-variate dataset. Every model has to detect any of the 4 possible defects out of the background. . . . .	99
6.5	Average Precision results for several models on the multi-variate dataset where Target D is considered as background. Please note that Target C and D are very similar and therefore discriminating between them makes this task extremely hard; the Target Detector has a consistent drop in performance whereas our deep learning approach suffers only a minor degradation. . . . .	100
8.1	Summary of the experiments and datasets. . . . .	125
8.2	Unimodal training and retrieval experiment on the CIFAR10 dataset. NN hash was trained on single modality only. Performance is shown as mAP in %. . . . .	127
8.3	Unimodal and cross-modal retrieval experiment on the CIFAR10 dataset. All methods were trained using multimodal data. CCA produces Euclidean embeddings. Performance is shown as mAP in %. . . . .	127
8.4	Unimodal training and retrieval experiment on the NUS dataset. NN hash was trained on single modality only. Performance is shown as mAP@10 / mAP in %, (– indicates no convergence was reached). . . . .	129
8.5	Unimodal and cross-modal retrieval experiment on the NUS dataset. All methods were trained using multimodal data. CCA produces Euclidean embeddings. Performance is shown as mAP@10 / mAP in %. . . . .	130

8.6	Cross-modal retrieval experiment on the Wiki dataset using 32-bit hashes (L2 with 32 tanh units) and Euclidean embeddings from Rasiwasia et al. [2010] (marked with *).	134
-----	---	-----



# Chapter 1

## Introduction

Recognizing and localizing objects is a crucial task in many computer vision, pattern recognition and industrial applications. If we look at the steel industry for example, a common application is to automatically detect defects in the material, based on camera images taken during the manufacturing process. Figure 1.1-(a) shows an image of a piece of defective steel. In this context, the problem involves solving the two following tasks: (1) segment the defective region from the background, and (2) classify the defect in the image, if any is present. These two tasks, while being at two different levels of abstraction, have common roots in feature extraction algorithms, which are able to describe the local or global structure of the input pixels by a vector of numbers, usually called a descriptor or, simply, *feature*.

Looking at Figure 1.1-(a) one may be tempted to design an ad-hoc algorithm to perform the task. In fact, for this particular case, taking the horizontal image gradient, and applying a threshold will do just fine. However, looking at Figure 1.1-(b), where the defect is in the form of a constellation of tiny dots, it becomes clear that such a direct approach is not trivial for the general case, especially when highly discriminative features are required so that efficient, linear classifiers can be used. The difficulty of this problem is mainly attributed to high intra-class variability (within the same class defects instances vary greatly) and inter-class variability (apparently similar defects belong to different classes). Of course, the same applies to general object recognition and detection. For example, in the case of digit recognition the same class is represented by a broad range of different instances, e.g. there are many ways to write a “3” and many of those ways make it hard to distinguish a “3” from an “8” or a “9”.

In the steel industry the process of hand-crafting features for real-time inspection systems has matured over the years, achieving exceptional performance

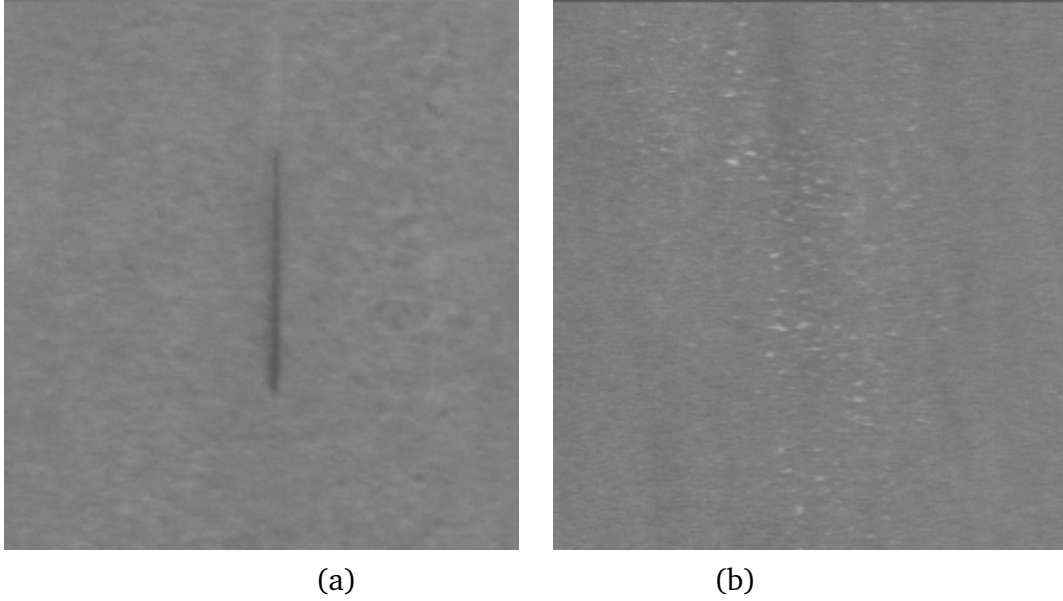


Figure 1.1. Sample images of two steel defects to illustrate the difficulties of the problem. On the left we have an extremely easy case, the defect is a vertical line, whereas on the right a challenging one, the defect is composed by the scattered spots. These images are taken from the current system and we clearly see that a big portion of the defect on the right is missing.

in many cases. Unfortunately, this is done by trial and error, which can take an enormous amount of time (e.g. current steel production systems are the result of more than 10 years of intense study), and seems to have already reached a plateau in terms of performance. Moreover, the modern steel industry moves quickly, so that the algorithms it uses to ensure quality must adapt at the same pace.

In particular, there are two ongoing developments which are problematic for the current engineered system: (1) demand for several highly textured steel grades is fast increasing, and (2) gray-level acquisition cameras, standard in actual systems, are being replaced by more sophisticated high-resolution cameras with infra-red, color and hyper-spectral sensors. The first requires a very large number of feature extractors to be generated because the number of defect classes increases rapidly as new grades of steel are introduced. Furthermore, these new grades tend to be much more refined so that it is a real challenge, even for an expert, to distinguish between defect types and between defect and background due to the high intra-class variability as seen in Figure 1.1-(b): bright



spots vary greatly in intensity and size, and can be spread over areas of several shapes that can vary by orders of magnitude.

The second development raises the challenge of feature extraction to the next level because now it must be performed on multi-variate images where domain knowledge has not yet been consolidated. For example, most of state-of-the-art computer vision systems rely on engineered features, such as SIFT [Lowe, 2004], which completely disregard any color information, and consider only a gray-level version of the input image. Even though there have been several attempts to extend these approaches to color (e.g. color-SIFT [Van De Sande et al., 2010]), and more generally to multichannel images, they have shown only marginal improvement on conventional vision tasks leaving the mainstream to their old-fashioned counterparts. While this may be reasonable for natural images, to some extent, it is not for steel – while a face can be recognized by its shape (using only edges) the only way to distinguish between many steel defects is to use additional spectra. In fact, certain defects are only visible in infra-red band.

The key limiting factor of all these hand-crafted approaches vis-a-vis these developments is that the features are fixed. For example, when we extract the SIFT descriptor from natural scenes or omni-directional images the same algorithm is used regardless of the different properties of these datasets (e.g. their probability distribution). Only expert knowledge, and extensive testing can determine which features perform best.

Given all these considerations, it has now become necessary to investigate alternatives to the standard engineered solutions that are easy to adapt and extend. Machine learning, and in particular *deep learning*, has in the past few years shown itself capable of learning low- mid- and high-level features for recognition [Jarrett et al., 2009; Zeiler et al., 2011]. The main advantage comes from the way deep learners compute features: multiple extraction stages (e.g. non-linear projections, neural networks) are chained together to form a deep architecture that is trained such that complex features are created in a bottom-up (hierarchical) fashion from the input data. Images, for example, are first decomposed into basic components (e.g. edges), later combined into corners and crosses and finally into object parts [Zeiler et al., 2011; Zeiler and Fergus, 2013b; Simonyan et al., 2013].

This multi-layer approach is in sharp contrast to engineered approaches which are inherently single-layer in structure, and are not amenable to stacking into hierarchies, and, therefore, cannot compose simple features into complex ones.

While deep learning can be understood also from a bio-inspired point of view, its popularity is due not to this appealing property, but rather to the fact that it

has recently achieved state-of-the-art results in interesting computer vision and pattern recognition benchmarks with [Ciresan et al. \[2011b\]](#); [Krizhevsky et al. \[2012\]](#); [Farabet et al. \[2013\]](#); [Ciresan et al. \[2012c,a,b\]](#), leaving competing approaches behind by a large margin.

Motivated by these recent achievements, this thesis advances deep learning models general and extends their application to the automatic surface inspection system (ASIS) of our industrial partner ArcelorMittal, a leading global steel company. Real industrial problems, in fact, are quite different from the sanitized environment of academic research, and off-the-shelf algorithms cannot be applied successfully without significant, domain-specific customization. The overarching approach is to develop new modules for steel quality control by formulating each particular processing task (i.e. stage in the processing pipeline) as a differentiable layer, or set of layers, that can be combined with others in a deep architecture, and trained together via gradient-descent (e.g. backpropagation). With recent advances in hardware, in particular the general purpose graphic processing unit computing (GPGPU), these models can be easily trained and tested in relatively short time.

Particular attention is devoted to the hot-strip mill section of steel industry. At this stage, the steel is in form of a plain, continuous sheet which is later rolled into coils ready to be shipped. The metal is visually inspected while sliding at considerable speed ( $\approx 800m/min$ ) to check for defects and anomalies. The production plant sets a grade for the steel quality and, according to the amount and type of the defects present, the production may be downgraded to a lower grade. Missing critical defects could hence bring to market a product which does not fulfill the requested specifications. This, of course, can be very expensive and can adversely effect on a company's reputation, in particular when expectations are high.

The contributions of this thesis can be split according to the time-scale in which they improve the performance of the processing pipeline. On the *long time-scale*, a series of deep learning algorithms are presented which will require changing the existing processing pipelines to improve segmentation and classification.

First, in **Chapter 3**, we establish that the currently most powerful deep architecture for vision, the max-pooling convolutional network (MPCNN), is indeed a viable and effective alternative to the current industrial feature extraction system. However, MPCNN are not directly applicable to steel defect classification because they cannot accomodate images of varying size, as is typical of steel inspection datasets. The first contribution of this thesis (**Chapter 4**) is a new architecture tailored to the general steel defect classification task that is free

from the input size constraint and offers new tools to boost the performance of MPCNN in this challenging application scenario. With these advances, much more targeted, task-specific features can now be *learned* and applied efficiently to these problems.

In principle, MPCNN can be used for image segmentation and object detection as well, but, in practice current architectures are too computationally expensive. To use them also for these tasks, we propose (**Chapter 5**) a new architecture which runs at speeds that approach the real-time requirements of industry. This is achieved by identifying and removing all redundant calculations from the segmentation process, and by devising a more efficient training procedure for MPCNN. In **Chapter 6**, this implementation is applied to the very challenging task of multivariate image segmentation.

On the *short time-scale*, the contributions focus on methods to improve the system that require almost no changes to the pipeline—this is crucial for industrial projects as even modest changes to the pipeline incur an lengthy testing phase before production can benefit from them.

Even for old and consolidated setups that use gray-level cameras, the installation of an ASIS system in a new production plant is a cumbersome task which can take up to several months because the entire set of parameters in the pipeline have to be adjusted to the new and slightly different environment. For the filtering section of the pipeline, mathematical morphology is widely used for its sophisticated, non-linear operators which deliver excellent performance, in particular for detection and image regularization, and because of its fast implementations. Tuning such operators, which often are composed by deep chains of simple ones, is a long process which requires expert knowledge and that is usually suboptimal. In **Chapter 7**, we consider this aspect in terms of improved filtering pipelines for detection and image enhancement (e.g. de-noising) by *learning* morphological operators. Several attempts have been made to learn such complex image processing tools Salembier [1992a,b]; Pessoa and Maragos [1998]; Wilson [1993]; Harvey and Marshall [1996]; Nakashizuka et al. [2010], however, they have mostly been limited to a particular operation or make strong assumptions which are sometimes unrealistic, especially for real scenarios. We instead tackled this problem from a deep learning, data driven, perspective and show that the learned operators match or outperform the classical hand-designed ones while being much more flexible.

Even when features are hand-designed, instead of learned, their effectiveness for segmentation and defect classification can still be greatly improved by embedding them into a new, *metric* space which enforces invariances (e.g. light conditions, camera etc.). For example, in the ASIS pipeline, when the number of

images that must be compared to is large (i.e. the database of defects), the nearest neighbor search that is normally used for classification can be prohibitively slow. By mapping image features into a metric space where relevant results for a given query are *clustered together*, search can be conducted much more efficiently. This can be accomplished through similarity sensitive hashing, but, as a linear method, it can perform poorly and cannot cope with the more general problem of measuring similarity between objects across different modalities, e.g. images and text.

In steel industry, leading companies such as ArcelorMittal have collected very large datasets of labeled defect samples that are used to tune a new production plant installation. Unfortunately, many of these images do not share the same features because they were acquired with different systems or the algorithms used to produce the image representations may have changed over time. An open problem, therefore, is how to exploit this heterogenous and costly data in order to improve classification accuracy and reduce the setup time for new production plants. **Chapter 8** formulates this problem as a multimodal similarity problem and introduces a neural network architecture, the *coupled siamese network*, that is able to learn a non-linear representation where images from different sources (e.g. modalities), are mutually comparable. This not only enables search in this highly heterogenous setting, but makes it more accurate, and an order of magnitude faster while requiring only a tiny fraction of the original storage space. We are among the first to propose a model for this task.

The next chapter introduces the background concepts which are required to understand the work presented in the rest of this thesis.

# Chapter 2

## Background

In this chapter we introduce in details the industrial problem domain that is the focus of this thesis and motivate the choice of deep learning as a means to address its challenges.

We begin with the basic concepts behind automatic surface inspection systems and detail how several operations are currently performed in production scenarios along with the corresponding challenges and expected improvements from our industrial partner ArcelorMittal. We continue with an overview of deep learning. After a short historical digression we explain the basic foundation methods required to understand the main models of our study, the max-pooling convolutional network and the morphological network. These are the state-of-the-art architectures for vision and image processing tasks. A thorough discussion and an updated list of its variants is reported for a general and comprehensive treatment of the subject. This is followed by a pragmatic introduction to mathematical morphology and its application to the steel inspection system is given so that the unfamiliar reader can better understand our contribution in learning morphological operators.

We conclude with an introduction to supervised metric learning, a widely used technique to boost performance which has recently gained a great deal of attention. Our treatment is limited to the two most popular methods which can be cast as deep learning instances, namely Neighborhood Component Analysis and Siamese Networks. This latter class of models will be foundation of our last study where we show that the Hamming metric can be utilized in place of the conventional Euclidean one thus reducing retrieval time and space complexity.

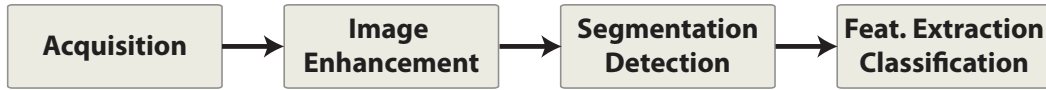


Figure 2.1. Illustrative example of a steel ASIS system processing pipeline.

## 2.1 Automatic Surface Inspection Systems

Automatic surface inspection systems (ASIS) are the key element of quality control in modern steel industry and general to many other industrial settings such as textile manufacturing and automotive, for example.

During steel production several defects can be generated and it is the job of the ASIS to automatically detect them and assess their priority given the current desired production grade. This process is the final stage before the product is delivered to the customer and therefore inaccuracy can deeply influence the company's reputation, and by direct consequence its profitability.

Automated systems are now standard even in small production plants because of their accuracy and speed. In fact, modern production schedules require quality control to be conducted at speeds that are far beyond human inspectors capabilities. Of course, human inspection is in many cases still required but should ideally be limited only to special cases, in other words a good system should have a very low number of false positives (e.g. raises an alarm when no defect is present) and leave the human inspectors to other more important and harder-to-automate tasks.

At a fairly high level of detail a standard ASIS can be decomposed into the following processing stages (schematic representation in Figure 2.1):

1. *Acquisition.* Usually gray-level linear cameras are utilized. This kind of cameras read the image line-by-line forming a buffer of several hundred "slices" which are then converted in an image. Compared to normal matrix cameras, which acquire the entire image at once, they are usually much more robust to sudden changes in illumination. Matrix cameras are sometimes utilized as well since they may offer acquisition features which are not present in linear systems, because of their cost, or simply because the software is designed for them.

A very important aspect of this stage is the lighting source; carefully chosen lights are in fact paramount to better capture particular kind of defects. Having optimal acquisition conditions in real production settings is

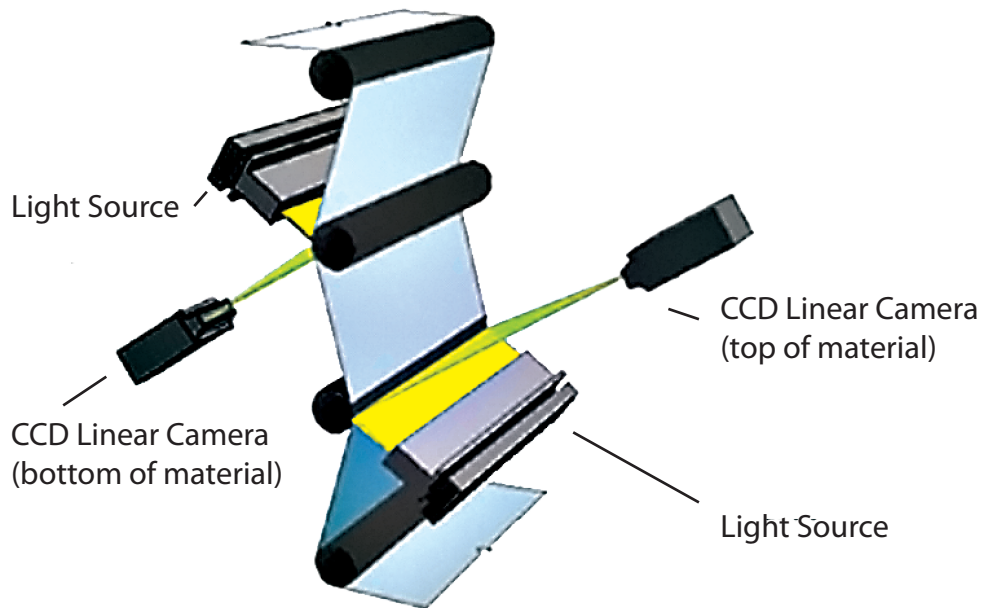


Figure 2.2. Illustrative example of a steel ASIS acquisition system. Source: ArcelorMittal. The steel is inspected from both sides by linear cameras and carefully chosen light sources.

of course hard to achieve and often images are out of focus, over- or under-exposed, mainly because of high temperatures and extreme environmental conditions.

As exemplified in Figure 2.2 images are captured for both the superior and inferior part of the steel coil to ensure that it is entirely covered and that eventual defects are captured on both sides of the material.

2. *Image enhancement.* Image processing is applied to ameliorate the acquisition, remove noise and enhance given structures in the image (e.g. the defect). At this stage, a widely used technology is mathematical morphology because of its fast implementation on dedicated chips, and because of its powerful operators which can be easily used to suppress the background and enhance the defects for an easier processing at the subsequent stages.
3. *Segmentation.* The “cleaned” images are then segmented into foreground and background, aiming at extracting only plausible anomalous areas. At

this stage a preliminary feature extraction may be performed as well to ease the process in discarding the largest portion of the non-defected areas in the image. The conventional approach for such task involves a series of linear and non-linear filtering operators followed by thresholding. The output of the segmentation stage can have several forms. Every input image can be transformed into a binary image, where each pixel is assigned either to the background or to the defect, or it can produce more elaborated outputs such as agglomerates of pixels as in super-pixels [Felzenszwalb and Huttenlocher, 2004; Achanta et al., 2010], areas and flat-regions [Salembier Clairon and Wilkinson, 2010; Crespo et al., 1997].

4. *Detection.* Given this segmentation output few additional features are computed to better characterize the various image parts so that a region-of-interest (ROI) can be extracted and the corresponding patch tiled from it. As a defect can span many slices of the strip this process has to take into account also this possibility before tiling the patch.
5. *Feature extraction.* Given an ROI a set of characteristic features is computed, which usually considers local and global properties of the image such as shape, area, elongation, gradient, texture, and so on. The concatenation of all these features produces a vector of numbers, the *image descriptor*.
6. *Classification.* The descriptor is classified into one out of the possible defect categories and human inspectors, if needed, make decisions on whether the current roll meets the required standards or it should be downgraded to a lower quality level or, even worse, discarded.

Advances in acquisition systems in the past few years (e.g. larger sensors and higher definition) have made it necessary for the algorithms to operate on large amount of data in real-time. Most pipelines have resorted to dedicated implementations to cope with this and fortunately hardware has more than kept pace. Nevertheless, algorithms should be computationally efficient and should not rely only on hardware advances for their applicability. Computationally efficient algorithms can easily be implemented, require cheaper hardware and are also cheap to run (e.g. require less energy).

From a machine learning perspective we would be, in principle, interested in all stages of the aforementioned ASIS pipeline where parameterizations could be learned to improve some quality criterion. In practice we focus on stages from 3 onward because of the high industrial interest and because they are not directly



related with a particular camera or hardware device which impose severe limits on what can be done.

It is only at the very last stage that some basic machine learning techniques have so far been applied to learn a supervised classifier. The preferred choice for the classifier is, however, a variant of  $k$ -nearest neighbors which involves almost no learning. This is because it is easy to add and remove samples without retraining and because it makes it easier to perform a visual validation of the system. For example, the user, given a new anomalous region, may simply decide to define a new class to the classifier by putting it in the defect database. Without going through retraining, the system immediately recognizes, at least in theory, the new class of defects. At the same time, the user can add images to known classes to try to improve the overall system performance. It is easy to see that this approach, while in principle appealing, has many drawbacks due to the high variability of defects and most importantly because of the non-adaptive feature extraction strategy (e.g. if the provided features do not discriminate between two classes of defects there is no way of adding images to improve the classification accuracy).

In the literature, perhaps because industrial applications tend to be patented or to be never disclosed, there is not much about steel defect detection [Martins et al., 2010]. However, in a broader context, the problem can be viewed as defect detection in textured material which has received considerable attention in computer vision [Leung and Malik, 2001; Varma and Zisserman, 2003, 2009]. In classical approaches, feature extraction is performed using the filter-bank paradigm. Each image is convolved with a set of two-dimensional filters, whose structure and support come from prior knowledge about the task, and the result of the convolutions (filter responses) is later used by standard classifiers. A popular choice for the two-dimensional filters are Gabor-Wavelets that offer many interesting properties and have been successfully applied for defect detection in textured materials [Kumar and Pang, 2002], for textile flaw detection [Bodnarova et al., 2002] and face recognition [Ayinde and Yang, 2002]. While a very powerful technique, it has many drawbacks. First of all it is inherently a single layer architecture whereas deep multi-layer architectures are capable of extracting more powerful features [Jarrett et al., 2009]. Furthermore, the filter response vector after the first layer is very high dimensional and requires further processing to be handled in real-time/memory-bounded systems.

### Why feature learning?

The current industrial system has often more than satisfactory results, product of years of expert knowledge. The main goal of our thesis work is to propose models which are able to obtain better performance via learning from the data with minimal prior knowledge. We want to leverage the burden which is still left to human inspectors and advance the state-of-the-art of ASIS systems. This has become a paramount research direction for industry as engineering new systems cannot keep the pace of the new hardware development. In particular we refer to new acquisition systems able to deliver multi-channel images and to new steel grades which make the task of hand designing to the extreme of human feasibility.

Additionally, such systems are never general enough to be used without cumbersome tuning processes in new environments. This means that, given a working and tuned system for a given production line, there is no good way of using it in a different production plant even though the production line might very well be the same. The process has to undergo several trial and error iterations and involves at least an expert in the field to perform such.

In contrast, we aim in producing a more general set of algorithms which can be employed with minimal setup times and minimal domain knowledge.

## 2.2 Deep Learning for Vision

Deep learning refers to a class of machine learning methods which produce the output after a long sequence of nonlinear transformations are applied to the input. Although its entrance into the mainstream of machine learning is quite recent, the origins of deep learning date back to the beginning of the artificial neural network era with the advent of the back-propagation (BP) algorithm. Many BP-like methods have been developed over the years [Bryson, 1961; Kelley, 1960; Dreyfus, 1962; Linnainmaa, 1970; Werbos, 1974; LeCun, 1987; Rumelhart et al., 1986].

This general idea of adding more than a single layer of nonlinearity to create more powerful models, has always driven neural network research, even though without falling under the name of deep learning. However, people have been struggling in training such architectures and, in conjunction with the limited computational capabilities of machines back then, other methods such as support vector machines (SVM) took over the pattern recognition scene.

It took almost twenty years and a list of outstanding results to bring back

neural networks as the model of choice in several applications ranging from image classification, segmentation and feature learning. At the core of all such results, and in sharp contrast to SVM, is the ability to learn features. This is particularly true for image classification where inputting the mere pixel representations makes no sense in most applications and a good set of image descriptors needs to be found.

The ultimate deep learning model is with no doubt the recurrent net [Werbos, 1974; Williams and Zipser, 1989; Schmidhuber, 1992; Graves, 2008], the preferred choice for sequence modeling in tasks such as speech recognition [Graves et al., 2006, 2013]. This architecture is in fact indefinitely deep by design and can work with arbitrarily long sequences; as a matter of fact it is Turing complete (refer to the PhD thesis of Alex Graves (2006) for a comprehensive treatment).

For the rest of the section, and throughout the rest of the thesis, we assume that we are given a dataset  $\mathcal{D} = \{(x_i, t_i)\}_{i=1}^N$  composed of  $N$  training pairs. Here  $x_i \in \mathbf{X} \subseteq \mathbb{R}^m$  represents the input patterns and  $y_i \in \mathbf{Y} \subseteq \mathbb{R}^k$  the target vectors. We also have minimization problem described by the following loss function

$$\theta^* = \underset{\theta}{\operatorname{argmin}} L(\xi(\mathbf{X}), \mathbf{Y}; \theta). \quad (2.1)$$

where  $\theta$  represents the whole set of parameters of the model  $\xi$ ,

Very briefly the objective of deep learning is to *find a reasonably good  $\theta^*$  with an arbitrarily complex  $\xi$  and a perhaps extremely large parameterization  $\theta$ .*

The intuition behind this is that by means of  $\xi$ , which operates directly on the input data, we aim in disentangling the explanatory constituent of it, therefore discovering what is really important for the given task. For example, if we want to have a face detector,  $\xi$  will have to remove task irrelevant information in the image such as the background, clothing, colors and focus on eyes, hair and mouth. Hence, the input image is decomposed in several factors so that what is *important* can be easily captured.

It is paramount to ease the application of machine learning methods to be able to discover such data representation with as less human intervention as possible. This way, independently on the input data, after feature learning all conventional machine learning methods will be applicable. SVMs for example suffer the feature extraction step; if you have good features they will produce robust classification results but if you lack the features, or if the features are poor, they wont be any good.

This explaining away factor drives novel feature learning algorithms among which supervised deep learning models, where this stage can be combined with the classification, excel in a broad and diverse range of applications. The best of

the representations in case of classification is of course the one to which a linear classifier is enough to get state-of-the-art performance. A remark that the devil is in the *features* [Chatfield et al., 2011].

In what follows we review the basic class of models which populated deep learning in the past years from a high level perspective. We start from the simplest linear model, the basic constituent, and from there build the multilayer perceptron. We finally dedicate a large section to convolutional neural networks, our model of choice for image classification from which we will derive the main contributions of this PhD study.

### 2.2.1 Linear regression and multilayer perceptron

Let us start with the linear regression model, a single-layer architecture that forms the foundation of deep learning. In this case the set of parameters,  $\theta$ , is represented by a linear projection  $\mathbf{W} \in \mathbb{R}^{m+1 \times k}$ , where the additional dimension is for the bias (a fixed input dimension set to 1 is added). The model computes the following function

$$\xi(x_i) = x_i \mathbf{W}, \quad (2.2)$$

and minimizes an instance of the loss function in eq. 2.1 given by

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^N \frac{1}{2} \|\xi(x_i) - y_i\|_2^2 + \lambda \Omega(\mathbf{W}), \quad (2.3)$$

where  $\Omega$  is a regularization function and  $\lambda$  its weight relative to the data term.

In case of  $\lambda = 0$  the problem reduces to least-squares and has analytical solution given by multiplying  $\mathbf{Y}$  by the pseudo-inverse of  $\mathbf{X}$

$$\mathbf{W}^* = \mathbf{X}^\dagger \mathbf{Y}. \quad (2.4)$$

A common regularization term is the so called ridge-regression (also known as Tikhonov-regularization) where  $\Omega$  enforces smooth solutions via

$$\Omega(\mathbf{W}) = \sum_{i=1}^m \sum_{j=1}^k w_{(i,j)}^2. \quad (2.5)$$

This formulation also has a closed form solution:

$$\theta^* = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{Y}. \quad (2.6)$$

Of course it is not always possible to obtain an analytical solution for this problem and often we have to resort to iterative optimization schemes.

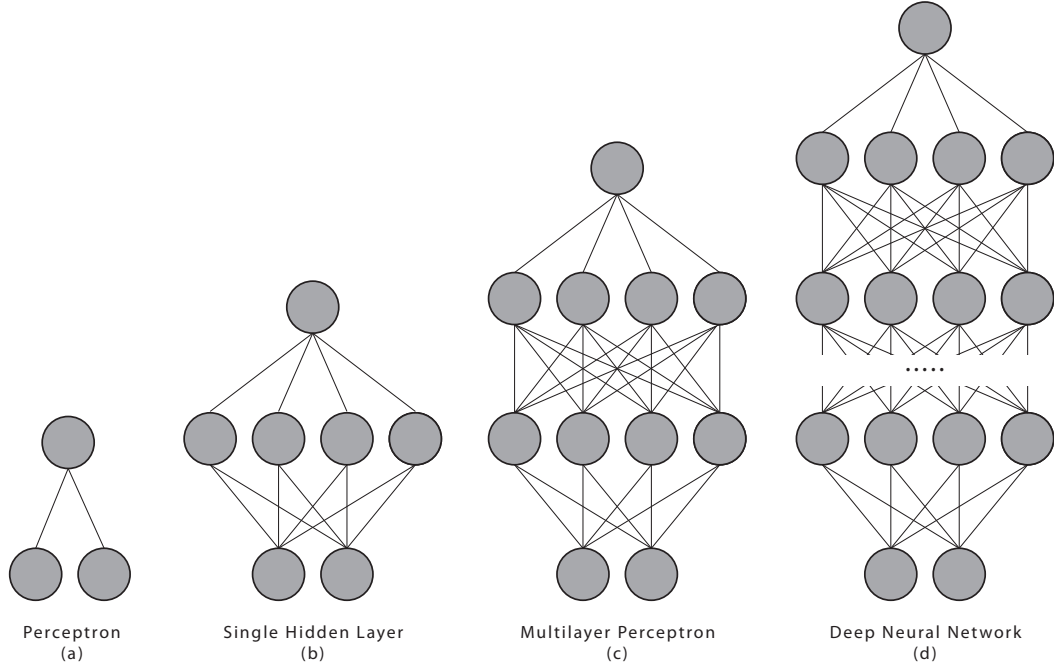


Figure 2.3. Illustrative example of several models which made the history of deep learning. We go from a single layer perceptron to the multilayer perceptron and reach the current state-of-the-art of deep networks which have many more layers of nonlinearity than conventional multilayer perceptrons.

In case of binary classification,  $\xi$  should produce a prediction on the membership of the given input  $x_i$  to one of the two possible classes, for which the ground-truth is available in  $y_i$ . The first neural network to ever do this was the *perceptron* [Rosenblatt, 1957] (see Figure 2.3-(a) for a schematic representation). It computes the following function

$$\xi(x) = \begin{cases} 1 & \text{if } x_i \mathbf{W} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.7)$$

The projection,  $\mathbf{W}$ , which correctly classifies the data can be *trained* using the *delta rule* or backpropagation.

The perceptron described in eq. 2.7 is not very powerful as it can handle only linearly separable data (e.g. fails in solving the simple XOR task). Fortunately it can be extended with intermediate layers of nonlinear hidden units as shown in Figure 2.3-(b) -(c). A simple single layer model can therefore be extended to a deep neural network named, because of this stacking, multilayer perceptron and

firstly presented by [Rosenblatt \[1962\]](#). It is important that every layer undergoes some nonlinear operation such as

$$\xi(x_i, \mathbf{W}) = \sigma(x_i \mathbf{W}), \quad (2.8)$$

where  $\sigma$  is a nonlinear function such as the s-shaped logistic function

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad (2.9)$$

or the hyperbolic tangent

$$\sigma(x) = \frac{\sinh x}{\cosh x} = \frac{1 - e^{-2x}}{1 + e^{-2x}}, \quad (2.10)$$

because otherwise we would end up with a system which can be expressed by a single linear projection. In fact the composition of several linear operators  $x \dots \mathbf{W}_0 \mathbf{W}_{k-1} \mathbf{W}_k$  is simply equal to  $\hat{\mathbf{W}}x$  where  $\hat{\mathbf{W}}$  is obtained by pre-multiplying all the matrices.

The final model is the result of the composition of such nonlinear differentiable blocks, better known as layers,  $\xi^j(x_i^j, \mathbf{W}^j)$ , where the superscript indicates the particular stage of transformation (e.g. layer index) and its corresponding input and parameterization.

Because the operation of every layer is differentiable, or at least has sub-gradient, every layer can be trained with gradient descent by simple application of the chain rule of derivatives which produces the two following results, and that in the neural network jargon is called backpropagation.

$$\delta^j = \frac{\partial L(\xi(x_i), y; \theta)}{\partial x_i^j} = (\sigma'(x_i^j \mathbf{W}^j) \circ \delta^{j+1})(\mathbf{W}^j)^T \quad (2.11)$$

for the partial derivative of the loss w.r.t. the layer's input and

$$\frac{\partial L(\xi(x_i), y; \theta)}{\partial \mathbf{W}^j} = (x_i^j)^T \delta^{j+1} \quad (2.12)$$

for computing the gradient w.r.t. the layer's weights. Each layer computes the gradient w.r.t the input to allow layers that precede it to compute their gradient until the input layer is reached, therefore the name backpropagation as it follows the same path through the net, but in reverse order. The term  $\delta^j$  indicates the partial result of differentiation, which in the case of the mean-squared loss, is set as follows

$$\delta^{out} = \frac{\partial \frac{1}{2}(\xi(x_i) - y)^2}{\partial \xi(x_i)} = \xi(x_i) - y_i.$$

Starting from this initialization, coming from the output layer back to the input, all other partial results and gradient can be computed with a cost which is equal to the one of forward propagation. Please note that when there is only a single layer, backpropagation stops at  $\delta^{out}$  and reduces to the delta rule of the perceptron. For the details on how the back-propagation algorithm works please refer to [Bishop \[2006\]](#).

The multilayer perceptron is the typical example of how deep neural networks can be constructed. Several layers of nonlinearities, each of which allows backpropagation of the gradient, are stacked to form a hierarchy of increasing complexity (e.g. lower layers learn simple things whereas deeper layers learn complex structures in the data). Only recently we have started experiencing deep neural networks with more than 8 layers. Historically, many layers are hard to optimize and only few attempts managed to train successfully. Thanks to efficient GPU implementations able to train on massive amount of data, a deep multilayer perceptron, never acclaimed as best model for digit recognition, achieved state-of-the-art results [[Ciresan et al., 2010](#)] on the MNIST [[LeCun et al., 1998](#)] benchmark. Key elements are data augmentation (e.g. a technique to generate new digits from the ones in the training set, to prevent overfitting) and plain backpropagation on a GPU board (graphic processing units are graphic cards capable to offer high computational resources for cheap price).

The choice of the basic building blocks to create such models is key element and strongly characterizes the field of deep learning. For example, the model the we are going to present shortly, the convolutional network, is a multilayer perceptron where each block performs operations which are tailored for image feature extraction and whose parameters can be learned.

### 2.2.2 Convolutional neural networks

Convolutional neural networks (CNN or convnets) are hierarchical models that alternate between two basic operations, convolution and subsampling, reminiscent of simple and complex cells in the primary visual cortex [[Hubel and Wiesel, 1968](#)]; and visually schematized in Figure 2.4.

This convolutional structure is the foundation of many image processing and biological models, ranging from wavelet decomposition to visual cortex simulations. After the seminal work of [Hubel and Wiesel \[1968\]](#) and [Olshausen and Field \[1996\]](#) it is well accepted that the V1 region of the cortex performs on-center off-surround filtering, also known as excitatory/inhibitory behavior. Same behavior has been found to emerge also from the work of [Hochreiter and Schmidhuber \[1999\]](#) where the authors proposed a method which explicitly

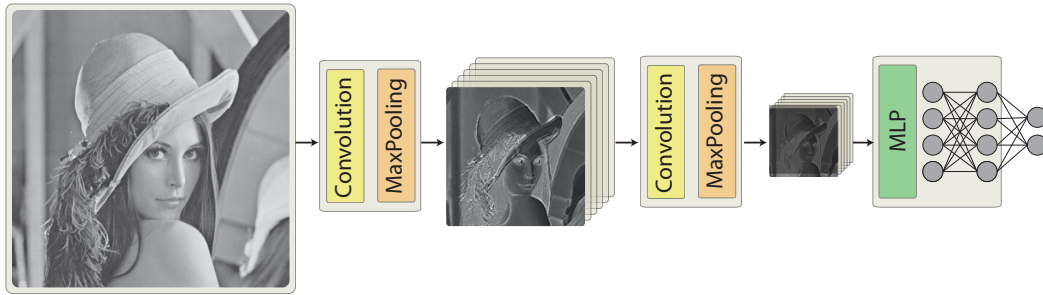


Figure 2.4. A schematic representation of an MPCNN; the process flows left to right. Raw input pixel values are processed by a number of interleaved convolutional and max-pooling (MP) layers, which are trained to extract meaningful features. Several fully-connected layers (MLP) follow, which produce the final classification.

takes into account the information-theoretic complexity of the code generator. The main characteristic of convnets is that they share weights, a small number of parameters sensitive to sub-regions of the high dimensional input signal (e.g. the image). These parameters are also called receptive fields, or convolutional kernels, and are replicated to cover entirely the visual field. Therefore, they are an excellent model for local feature extraction, the best known strategy in image processing. They find their origin in the pioneering work of Fukushima [1979, 1980] who first introduced them and in the work of LeCun et al. [1989a, 1990] who made them work with backpropagation and made them popular.

Because convnets share weights, the number of free parameters does not grow in proportion to the input dimensions as in standard multi-layer networks. Thus, they scale well to real-sized images and excel in many object recognition [Ciresan et al., 2011a,b; Ciresan et al., 2011; Krizhevsky et al., 2012; Sermanet et al., 2013b,a] and segmentation/detection [Ciresan et al., 2013a, 2012a; Masci et al., 2013b] benchmarks.

The original formulation of CNN, which can be found in Fukushima [1979, 1980] and LeCun et al. [1989a, 1990], has been considerably enriched during the years by new layers that helped in boosting its performance.

We use GPU-based deep convolutional architectures [Ciresan et al., 2011a] with max-pooling (MP) layers [Riesenhuber and Poggio, 1999; Scherer et al., 2010] instead of Fukushima’s alternative winner-take-all layers and, in the style of LeCun et al. [1989a] and Ranzato et al. [2007c], train them with backpropagation [Linnainmaa, 1970; Werbos, 1981]. The resulting architecture is named



**MPCNN.**

Since 1989, LeCun's lab has invented many improvements and simplifications of such CNN.

The following is an up-to-date list of the types of layers which are the most useful to perform image classification and segmentation and that best characterize latest advances in the field.

**Convolutional:** performs a 2D filtering between input images  $x$  and a bank of filters  $w$ , producing another set of images  $h$ . A connection table  $CT$  indicates the input-output correspondences, filter responses from inputs connected to the same output image are linearly combined. Each row in  $CT$  is a connection and has the following semantics: (inputImage, filterId, outputImage). This layer performs the following mapping

$$h_j = \sum_{i,k \in CT_{i,k,j}} x_i * w_k \quad (2.13)$$

where  $*$  indicates the 2D valid convolution. Each filter  $w_k$  of a particular layer has the same size and defines, together with the size of the input, the size of the output images  $h_j$ . Then, a non-linear activation function (e.g. tanh, logistic, etc.) is applied to  $h$  just as for standard multi-layer networks.

The error back-propagation for a convolutional layer is given by

$$\delta_i = \sum_{k,j \in CT_{i,k,j}} \delta_j \star w_k \quad (2.14)$$

where  $\star$  indicates the 2D full correlation, equivalent to a convolution with a kernel flipped along both axes.

The gradient is computed as

$$\nabla w_k = x_i * \delta_j. \quad (2.15)$$

In equation 2.15 there is no summation because, since each kernel is only used once, there are no multiple images sharing the same kernel. If it were instead the case that each kernel is used several times then the gradient of each filter should be accumulated over every input-output pair of images it was used for.

This transform exhibits edge detection properties at the lower layers common to most of the engineered feature extraction methods whereas the higher level features exhibit different behaviors, ranging from object parts to more generic class/object detectors.



Figure 2.5. Examples of subtractive normalization on a conventional (left) and on a steel defect (right) image.

**Subtractive Normalization:** This layer normalizes the input image,  $x \in \mathbb{R}^3$ , by locally centering every pixel over a given spatial support  $w \in \mathbb{R}^{k \times k \times c}$  spanning all, or a subset of, the channels. Such layer can be efficiently implemented with a conventional convolution. Let  $w = \frac{1_{k^2 \times c}}{k^2 * c} \in \mathbb{R}^{k \times k \times c}$  be the averaging convolutional kernel, the resulting normalization is performed as follows:

$$\text{subnorm}(x) = x - x * w, \quad (2.16)$$

where the convolution has a fully connected connection table. The effect of this type of normalization is shown in Figure 2.5 for both a natural and a industrial image.

**Divisive Normalization:** This layer performs a more complex operation derived from bio-inspired models such as the one proposed by Pinto et al. [2009]. If the input image is normalized with the subtractive normalization layer, it estimates the local standard deviation and normalizes every location to have unit variance. Therefore, it can be considered as a whitening scheme for natural images as it filters out high frequencies. Also this layer can be efficiently implemented with plain convolutions using unitary kernels (e.g. all ones) and has the following formulation

$$\text{divnorm}(x) = x \circ \frac{1}{\sqrt{x^2 * w}}, \quad (2.17)$$

where  $\circ$  indicates the Hadamard (element-wise) product. An example of this layer's output when a support of  $9 \times 9$  for the kernel is used is shown in Figure 2.6



Figure 2.6. Examples of divisive normalization on a conventional (left) and on a steel defect (right) image.

**Contrast Normalization:** Very often the subtractive and normalization layer are chained together to make the contrast normalization or subtractive-divisive layer. It has been found that this kind of normalization is paramount in obtaining state-of-the-art results in deep multi-stage architectures for object recognition as reported in earlier work by [Jarrett et al. \[2009\]](#). An example of this layer's output is shown in Figure 2.7.

**Tiled Convolution:** This type of processing considers local neighbors of the input pixels but, contrary to a conventional convolutional layer, does not share weights. It has been used in large scale unsupervised models trained in a full parallel setting by [Le et al. \[2010, 2012\]](#). While sometimes is useful, in particular in obtaining state-of-the-art results on CIFAR10 [[Krizhevsky, 2010](#)] (e.g. locally connected layer in cuda-convnet [[Krizhevsky, 2011](#)]) when used at the very last stages, most of the proposed models prefer convolutional layers with shared weights.

**Laplacian Pyramid:** This is a type of transform which is very popular in computer vision. Just to give an example, this is the first stage of the SIFT [[Lowe, 2004](#)] descriptor and is used to produce features which are scale invariant. Given a set of Gaussian kernels  $w_{\sigma^i}$ , where  $\sigma^i \in \Sigma = \{\sigma^0, \sigma^1, \dots, \sigma^L\}$  is the corresponding standard deviation (i.e. the smoothing factor) and an input image  $x$  this layer produces its output as follows

$$laplacianpyr(x) = \{y_i | y_i = x * w_{\sigma^i} \text{ for } \sigma^i \in \Sigma\} \quad (2.18)$$



Figure 2.7. Examples of subtractive divisive normalization on a conventional (left) and on a steel defect (right) image.

This layer has been used in recent work of [Farabet et al. \[2013\]](#) to input the network with a representation more suited for a multi-scale processing and helped in obtaining state-of-the-art results for scene parsing and pedestrian detection by [Sermanet et al. \[2013b,a\]](#).

**Pooling:** A pooling layer reduces the dimensionality of the input by a constant factor along the width and height dimensions. The purpose of this layer is not only to reduce the computational burden, but also to perform feature selection. The input images are tiled into non-overlapping subregions from which only one output value is extracted. Common choices are maximum or average, usually shortened as max-pooling (or MP) and avg-pooling. Recently [Sermanet et al. \[2012\]](#) proposed a pooling based on the  $\ell_p$  norm of the neighborhood. Max-pooling is generally favorable as it introduces small invariance to translation and distortion, leads to faster convergence and better generalization [[Scherer et al., 2010](#)].

Figure 2.4 shows an example of a  $2 \times 2$  max-pooling operation, whose output illustrates the effect of this feature selection step where only locally strong features are kept.

The back-propagation step of a max-pooling layer places the  $\delta_i$  value for the region in the location where the maximum value was found during the forward pass. In case of average-pooling, instead, such value is scaled by the number of elements in the pooling region and placed in the entire region.

**Fully Connected Layer (MLP):** this is the standard layer of a multi-layer network. It performs a linear combination of the input vector with a weight matrix. Either the network alternates convolutional and max-pooling layers such that, at some stage, a 1D feature vector is obtained (images of  $1 \times 1$ ), or the resulting images are rearranged to have 1D shape. The output layer is always a fully connected layer. In a classification task, for example, we have as many neurons as classes and outputs normalized with a softmax activation function to approximate the posterior class probabilities.

### 2.2.3 Learning strategies

Deep models, as described here, can be trained with gradient techniques and therefore any algorithm such as conjugate gradient and newton methods can be used. However, as the number of free parameters in these models is very large (e.g. order of millions), and also the number of training samples is usually extremely large, online methods are preferred. Otherwise a complete sweep through the dataset would be required to perform just a single model update. Batch techniques, in particular, are inapplicable when data augmentation, virtually producing an infinite dataset, and paramount to obtain state-of-the-art performance, is employed.

Over the years stochastic gradient descent has established itself as the deep learning algorithm of choice. More interestingly, recent studies showed that online learning can also approach the convergence rate of much more complex methods while still retaining the good properties of the stochastic updates [Xu, 2011; Bottou, 2010].

The pseudocode, with Matlab-like notation, of the stochastic gradient descent algorithm is shown in Algorithm 1. The initial learning rate  $\eta$  is annealed by a factor of  $\alpha$  after every epoch has been completed. A momentum term  $\mu$  is also included as it is paramount to escape local minima and to improve convergence. The additional parameter to be chosen is the mini-batch size,  $bs$ , which indicates how many randomly selected samples should be used to compute the estimate of the gradient over the entire dataset  $\mathcal{D}$ .

A sometimes very effective way of training deep models, is via a hybrid mix between batch and online algorithms as proposed in Ngiam et al. [2011]. The idea is to sample a fairly small number of data points from the training set and perform a small number of updates, usually ranging from 5 to 10, of a second order algorithm such as LBFG-S and iterate until convergence.

---

**Algorithm 1:** Pseudocode for the stochastic gradient descent method with learning rate annealing ( $\alpha$ ) and momentum ( $\mu$ ).

---

**Data:** model  $nn$ , input  $\mathbf{X}$ , target  $\mathbf{Y}$ ,  $\eta$ ,  $\mu$ ,  $\alpha$ ,  $bs$ ,  $nE$   
**Result:** updated model parameters  $w$

```

 $w = nn \rightarrow getParams();$ 
 $dwold = zeros(length(w),1);$ 
 $nBatches = ceil(X.nSamples / bs);$ 
for  $e=1$  to  $nE$  do
     $idxs = randperm(X.nSamples);$ 
     $cost = 0;$ 
    for  $i=1$  to  $nBatches$  do
         $from = (i - 1) * bs + 1;$ 
         $to = \min(i * bs, X.nSamples);$ 
         $[x, y] = nn \rightarrow getDataBatch(X, Y, idxs(from : to));$ 
         $[cost\_b, g] = nn \rightarrow grad(w, x, y);$ 
         $cost = cost + cost\_b;$ 
         $dw = \mu * dwold - \eta * (g / (to - from + 1));$ 
         $w = w + dw;$ 
         $dwold = dw;$ 
    end
     $\eta = \eta * \alpha ;$ 
end

```

---

### Model initialization

When dealing with several layers of nonlinearities it is not trivial determining how to properly initialize the model's parameters so that the optimization delivers a better solution and does not get stuck in a bad local minima. While lot of heuristics have been developed to address such issue there is no clear answer to this problem [[Larochelle et al., 2009](#); [Glorot and Bengio, 2010](#)].

Another very popular and effective technique to initialize deep neural network models is instead based on a less heuristic approach and uses the weights of a trained unsupervised model as initialization. One of the preferred models for such task is the autoencoder.

**Autoencoders** are unsupervised models mapped as feedforward shallow nets (single or at maximum two layers only are used) trained to reconstruct their



input. The general loss is given by

$$L(\mathbf{X}; \theta) = \|\xi(\mathbf{X}) - \mathbf{X}\|_2^2 + \lambda\Omega(\xi(\mathbf{X})) \quad (2.19)$$

where the first term measures the reconstruction error, or data-fit, and the second acts as a regularizer;  $\xi$  indicates an arbitrary network parameterized by  $\theta$ . Common choice for the regularizer is given by the  $\ell_1$  norm of the network's activations,  $\xi(\mathbf{X})$ , as it helps in disentangling the input patterns getting only the underlying regularities.

While in theory  $\xi$  could be composed by several layers, minimizing the loss of eq. 2.19 in such situation has been proven to be extremely hard and therefore avoided. Because of this, the conventional learning setup is to use a single layer network and stack several models training each of them, bottom-up, on the output of the model below; exception given for the first one which takes the actual input.

The single layer autoencoder formulation is as follows

$$h = \sigma(\mathbf{XW}) \quad (2.20)$$

$$y = h\mathbf{W}^T \quad (2.21)$$

when the same set of parameters (*tied weights*) is used both for encoding and decoding. Other variants have different sets of weights and are named *untied*. In case of a linear activation and tied weights  $\sigma(\mathbf{X}) = \mathbf{X}$  we obtain PCA. For a more detailed description of unsupervised models we refer to the excellent work of the PhD thesis of [Ranzato \[2009\]](#) and [Kavukcuoglu \[2011\]](#).

Things get more interesting, of course, when the mapping is nonlinear and more complex structures in the input patterns can be discovered. In such cases one of the best ways to regularize the model is by means of the Jacobian of the output, or reconstruction, with respect to the input as in the work on contractive autoencoders of [Rifai et al. \[2011\]](#). The regularization term  $\Omega$ , for a given input pattern  $x$ , takes the following form

$$\|J_\xi(x)\|_F^2 = \sum_{i,j} \left( \frac{\partial h_j(x)}{\partial x_i} \right)^2$$

which encourages the feature space (latent representation of the autoencoder) to be contractive in the neighborhood of the training data. The low valued first derivative induces invariance and robustness of the representation to small changes of the input.

**Layer-wise stacking** Once an autoencoder has been trained on the input data its hidden representation is used as input for another model. The process is iterated until all layers are trained in such layer-wise fashion [Hinton and Salakhutdinov, 2006; Bengio et al., 2012].

Then, the encoding only sections of every model are kept therefore producing a deep neural network, as the one in Figure 2.3-(d), whose weights have been initialized through unsupervised learning. Such an approach has been found to excel in avoiding bad local minima and produces models which more often hit a “good” spot as reported by the extensive evaluation of Erhan et al. [2010].

Following the same line of works several other approaches have been presented both for feed-forward networks and for convolutional architectures [Masci et al., 2011a; Lee et al., 2009b; Ranzato et al., 2007a; Zeiler et al., 2010; Coates et al., 2010]. While being very popular and appealing, unsupervised pre-training requires additional time and it is often not clear whether it really helps in improving performance for tasks such as classification [Rigamonti et al., 2011].

Additionally, with the day-by-day increasing number of samples and recently developed techniques to prevent over-fitting, most of the approaches tend to avoid this stage.

#### 2.2.4 Preventing over-fitting

Deep models, approaching the billion of free parameters, are prone to over-fit (e.g. learning the training data because of the way too large representational power), therefore delivering poor generalization performance. The main reason resides in the lack of available training data to properly shape the model. Fortunately this issue has nowadays become less important; in fact on the one hand we have way larger training datasets approaching web-scale dimensions and on the other hand because very effective techniques to prevent over-fitting have been proposed. In what follows we give a short summary of the most popular approaches.

**Data augmentation** Virtually increases the amount of labeled training samples. Techniques range from additive noise to the input patterns to complex transformations which mimic data samples drawn from the same distribution. In the context of image classification with convolutional networks the most powerful data augmentation protocol is perhaps the one of Ciresan et al. [2011b] refined from the earlier work of Simard et al. [2003] which mixes elastic distortions with rotations and translations of the input images. Our system was the



first deep learning system to outperform human performance in the IJCNN11 traffic sign competition [Ciresan et al., 2012b].

Two other popular techniques use binomial noise for the network activation units or for the connections, respectively named as Dropout [Krizhevsky et al., 2012] and Dropconnect [Wan et al., 2013]. These are usually interpreted from a model averaging perspective but they nevertheless add noise to the model to prevent over-fitting. Especially when applied to the input layer they are with no doubt data augmentation techniques.

**Weights prior** It is a very popular technique to add prior to the parameters of the model in order to find low-complexity networks capable of better generalization. We have seen already one of such techniques for eq. 2.2 where  $\Omega$  enforced small weights through weight decay [Hanson and Pratt, 1988; Rurnelhart and Huberrnan] whose formulation can be derived by Gaussian or Laplace priors [Hinton and van Camp, 1993].

Other approaches have followed a different path and generated the network in a sequential fashion [Fahlman, 1991; Ash, 1989], pruned the input [Moody, 1992] or the network's units [Mozer and Smolensky, 1989].

Similarly, “Optimal Brain Damage” of LeCun et al. [1989b] proposes a way to remove unimportant connections to adapt the size of the neural network in a practical and sound manner.

Schmidhuber [1997] showed that using theory of algorithmic complexity networks could be regularized to achieve generalization performance unmatched by more traditional algorithms. The key is to favor solutions computable by short and fast programs.

**Unsupervised pretraining** The most common approach takes an autoencoder and uses it to initialize the weights as explained in the previous section. It has been also shown that training to de-noise input patterns where some features are switched off (binomial noise) or where gaussian noise is added can produce representations which are much more robust and effective as pre-training stages [Vincent et al., 2008; Bengio et al., 2013; Masci et al., 2011a; Vincent et al., 2010].

To summarize the so far most successful methods work as follows: 1) for each layer trains an autoencoder with the same structure as the layer in the deep model to pre-train; 2) discard the decoding (reconstruction) part and keep only the feed-forward section; 3) compute the activations and use them as input for the next layer.

**Sparsity** Another very popular approach, which follows the same idea of unsupervised learning, but using sparsity in the model; e.g. select a sub-model which is good enough to represent a particular data sample without exploiting the whole representational power. Initially developed as regularization for denoising and inverse problems [Elad, 2010] it has become widely used in deep learning as well.

The main optimization of sparse models is the “pursuit process”, which can be defined as

$$\operatorname{argmin}_z \|\mathbf{D}z - x\|_2 + \lambda\|z\|_1, \quad (2.22)$$

where  $\mathbf{D}$  represents a dictionary of basis to represent a given input signal  $x$ . As the number of basis is usually larger than the number of elements in  $x$ , e.g. overcomplete, the problem has no unique solution. The  $\ell_1$  penalty is therefore added to discriminately select a small subset of the dictionary to be used, along with the vector of coefficients  $z$ , for explaining the input data  $x$ . Please note that we directly optimize over  $z$  rather than over  $\mathbf{D}$  as with autoencoders. Moreover, the hidden representation of an autoencoder is obtained with a single projection and does not undergo an explicit optimization as stated in eq. 2.22.

Many interesting problems follow the same formulation and are efficiently solved with FISTA (Fast Iterative Shrinkage Thresholding Algorithm) [Beck and Teboulle, 2009] and ADMM (Alternating Direction Method of Multipliers) [Goldstein and Osher, 2009] methods, just to cite the most notables.

As  $\mathbf{D}$  should ideally be learned as well, the problem is decomposed in two subproblems: dictionary learning and sparse code inference. The first optimizes over the dictionary,  $\mathbf{D}$ , so that it can better represent the data with very low reconstruction error. The second problem finds, given the dictionary, the sparse code. Optimization alternates between these two stages until convergence is reached.

Due to their success, sparse priors opened the path for a long list of related works which all deeply follow to the idea underlying sparse coding, but whose application is developed for deep learning models [Le et al., 2010; Lee et al., 2008; Ranzato et al., 2006a, 2007b; Ranzato and LeCun, 2007; Zeiler et al., 2011; Vincent et al., 2008; Bengio et al., 2007].

Most notably, the predictive sparse decomposition model of Kavukcuoglu et al. (2008; 2010; 2011), PSD for short, has gained lot of attention. In fact, while previous works were simply applying the idea of sparsity as regularization, they never aimed at obtaining at inference time a sparse code which approached the optimal one obtainable via an exact iterative pursuit process, such as FISTA, but with a fixed complexity (e.g. fixed number of iterations). The authors of

PSD trained a feed-forward network to approximate this exact solution without resorting in the computationally expensive iterative process which would otherwise be required. Results on challenging computer vision benchmarks showed the favorability of the approach versus previous ones confirming that an approximate solution is indeed desirable in practice. This is due by the unstable behavior of sparse codes; a small perturbation to the input may result in two largely different codes.

More recently, Sprechmann et al. (2012; 2013) proposed a form of recurrent neural network which goes a step forward. In fact, instead of learning an approximation of exact sparse codes as target vectors as in the PSD framework, the authors propose to cast the pursuit process as a neural network and to learn its parameters through gradient descent. The network is initialized as with the exact algorithm but its complexity, expressed as the number of iterations to be performed, is fixed. Their approach follows the original algorithm but learns the parameterization which better suits the given complexity and task; for example we can have purely discriminative models aimed at classification instead than reconstruction. Learning the pursuit process has several advantages and the work shows that just few iterations are enough to have a almost perfect approximation of the exact sparse codes which would otherwise require many more iterations.

**Stochastic model regularization** These techniques regularize the model by randomly injecting data agnostic noise (e.g. binomial or Gaussian) to its internal representation or parameters. While having been a very popular technique for years it has been only recently reinterpreted and brought to the mainstream of machine learning with the name of *Dropout* by Krizhevsky et al. [2012].

The general idea is to prevent features co-adaptation. In other words, features have to become statistically independent and "fire" independently on the others, therefore delivering the highest of the information throughputs. Another way of seeing it is as a model averaging technique. In fact, for any subset of active features we have a classifier (e.g. the subset of projections which are used), an exponential number of them. The output of the model is given by their geometric mean with a simple and efficient formulation; units during the test phase are not switched off and weights are rescaled to compensate for this. When softmax is used at the output layer, which is the case for classification problems, the final activation is already the geometric mean of all possible sub-models.

Let us now introduce in more details how this technique works. For a given activation set of a fully connected layer,  $a$ , dropout selects with probability  $p$  a random percentage of units at random and sets them to zero. This enforces

a non-deterministic fixed sparse representation which is unpredictable and that therefore makes features more robust; e.g. as useful as they can get. Every output unit is then kept with probability  $1 - p$ . Its matching function is the popular rectified linear unit (ReLU) defined as

$$\text{ReLU}(x) = \max(x, 0). \quad (2.23)$$

In fact, the output of a fully connected layer with weight matrix  $\mathbf{W}$  is given by

$$a = \sigma(\mathbf{W}x), \quad (2.24)$$

where  $\sigma$  represents the non-linearity.

As dropout sets to zero a portion of  $a$  we have

$$a = m \circ \sigma(\mathbf{W}x), \quad (2.25)$$

where  $m$  indicates the dropout mask.

In case of ReLU we have that  $\text{ReLU}(0) = 0$  and therefore

$$a = \sigma(m \circ \mathbf{W}x). \quad (2.26)$$

At inference time, during the forward propagation of the network at test time, we need to estimate the average output of a dropout layer over all possible masks  $M = \{m_0, m_1, \dots, m_{2^{|a|*p}}\}$ , which is obviously unfeasible. The authors proposed to approximate such computation with averaging before activation, which implements easily as scaling the learned weights by  $p$ .

Of course, training times are increased but the final result usually pays back for the additional training epochs. Dropout opened the path to several related contributions, here briefly reported:

- **Dropconnect** [Wan et al., 2013], is another way of regularizing fully connected layers by randomly switching off connections instead of units as in dropout. Its function is defined by

$$a = \sigma((m \circ W)x) \quad (2.27)$$

where  $m$  in this case is a mask for the connections. The authors emphasize the interpretation of their proposed model as mixture of experts similarly to dropout and give a perhaps more rigorous way of estimating their activation over all possible masks. The computation is in fact the weighted sum of Bernoulli variables and can be approximated by a Gaussian with moment matching. A very efficient and parsimonious implementation, as extension of the cuda conv-net library of Krizhevsky [2011], is also provided. Results shows superior, even though not outstanding, performance with respect to its dropout counterpart on a variety of challenging tasks.

- **Stochastic pooling** [Zeiler and Fergus, 2013a] replaces the deterministic max-pooling operator, used in all record breaking nets. Activations within each pooling region are pooled according to the multinomial distribution given by the local activity of the units in the pool. The proposed model is parameter free and the authors showed its superior performance on several datasets when no data augmentation techniques are used. The main motivation of the work is to propose an extension of dropout to convolutional layers, where this latter is not intuitively applicable from a mixture-model perspective but only in form of binomial noise. Also in this case, at test time, workarounds to avoid sampling an exponential number of configurations needs to be performed. In the paper it is proposed to use a simple weighted average which seems to work very nicely in practice.

**Winner-Take-All model regularization** Another popular form of regularization is based on the winner-take-all paradigm, used for the max-pooling layer for example, to operate as an activation function. The methods consider a local pool  $B$ , with fixed size  $|b|$ , of activations  $a$  and produce an output which is driven by local competition with the function  $\sigma$  within the group as follows

$$a^b = \sigma((Wx)^b), \quad (2.28)$$

where the superscript  $b$  indicates a given block.

- **MaxOut** [Goodfellow et al., 2013b]: such units expand the input dimensional space by a constant factor and emit the maxima value in the group. For convolutional layers it produces a number of intermediate maps which is  $|b|$  times larger than the desired output and pools along the maps-axis to shrink the representation down to the desired size. In case of fully connected layers it does a very similar thing, first expands the dimensionality of  $a$  and then it pools over  $B$ . In this case  $\sigma$  is the *max* function which returns a scalar value. MaxOut units are state-of-the-art in various benchmarks and a new landmark for further advances in this area thanks also to the availability of the code and of the experimental evaluation in pylearn2 [Goodfellow et al., 2013a].
- **Compete-to-compute**: we recently proposed [Srivastava et al., 2013] a regularization based on a similar approach which is also extremely easy to implement and does not require additional storage for the intermediate step as in MaxOut. In our case  $\sigma$  erases non maxima values in the given block, therefore not reducing but rather sparsifying the representation.

Similarly to dropout we obtain codes with a fixed sparsity. We experiment with block size of two for the fully connected layers. Our experimental evaluation shows that while being striking simple to implement our local competition scheme achieves state-of-the-art performance on digit recognition and sentiment analysis tasks. We also report interesting results for the catastrophic forgetting where our model retains useful representations, for one set of inputs, even after being retrained to classify another.

A similar approach was used in our previous work reported in Chapter 4. In that context we used a block of the same size as the hidden representation to mimic the hard assignment of the k-means coding scheme.

It is worth mentioning, however, that when the amount of training data is large, the benefit of these regularization techniques tends to vanish.

## 2.3 Mathematical morphology

Mathematical morphology (MM) is a nonlinear image processing methodology based on computing max/min filters in local neighborhoods defined by structuring elements [Najman and Talbot, 2013; Serra, 1982; Soille, 1999]. By concatenation of two basic operators, i.e., the dilation  $\delta_B(f)$  and the erosion  $\varepsilon_B(f)$ , on the image  $f$ , we obtain the closing  $\varphi_B(f) = \varepsilon_B(\delta_B(f))$  and the opening  $\gamma_B(f) = \delta_B(\varepsilon_B(f))$ , which are filters with scale-space properties and selective feature extraction skills according to the underlying structuring element  $B$ . Other more sophisticated filters are obtained by combinations of openings and closings, to address problems such as non-Gaussian denoising, image regularization, etc.

In what follows we briefly recall the most common morphological operators and their connection to steel industry. We finish with some challenges which will motivate our study on this topic. It is beyond the aim of this dissertation to give a complete overview of this vast field and we will introduce only concept that we will investigate in our study and that are highly related to steel industry. The interested reader can refer to Najman and Talbot [2013]; Serra [1982]; Soille [1999]; Santiago [2012]; Angulo [2009, 2003] for a comprehensive and detailed overview.

### 2.3.1 Morphological operators

Here only the basic gray-level operators are considered because those are the most interesting and applied in steel industry. Mathematical morphology is,

however, a very well developed and studied field which provides extensions to color and, more generally, multivariate images. Such treatment is beyond the scope of this manuscript and the interested reader can refer to the work of [Santiago \[2012\]](#).

Let  $f(x)$  be a 2D real-valued image, i.e.,  $f : \Omega \subset \mathbb{Z}^2 \rightarrow \mathbb{R}$ , where  $x \in \Omega$  denotes the coordinates of the pixel in the image domain  $\Omega$  and  $\mathbf{b} : \Omega \rightarrow \mathbb{R}$  is the fixed structuring function.

The further convention to avoid ambiguous expression is considered:  $f(x+h) + \mathbf{b}(h) = +\infty$  when  $f(x+h) = +\infty$  or  $\mathbf{b}(h) = -\infty$ , and that  $f(x-h) + \mathbf{b}(h) = -\infty$  when  $f(x-h) = -\infty$  or  $\mathbf{b}(h) = -\infty$ . A flat structuring function of the set  $SE$  (e.g. structuring element) is defined as

$$\mathbf{b}(x) = \begin{cases} 0 & \text{if } x \in SE \\ -\infty & \text{otherwise.} \end{cases} \quad (2.29)$$

With  $SE$  we indicate the subset of the discrete space  $\mathbb{Z}^2$  support of the image.

### Dilation and Erosion

Dilation and erosion are the foundation of every morphological filtering. Given an image  $f$  and a structuring function  $\mathbf{b}$  they are respectively defined as the convolution in the  $(\max, +)$ -algebra

$$\delta_b(f)(x) = \sup_{h \in SE} (f(x-h) + b(h)) \quad (2.30)$$

and its dual

$$\varepsilon_b(f)(x) = \inf_{h \in SE} (f(x+h) - b(h)) \quad (2.31)$$

Of particular interest for its developed theory and practical applications is the flat grey-level dilation and erosion which is obtained when the structuring function is flat and becomes a structuring element [\[Soille, 2003\]](#). A flat structuring element defines the shape of the structures of interest and can be considered as a indicator function.

Therefore we can redefine the dilation and erosion with a flat structuring element as

$$\begin{aligned} \delta_{SE}(f)(x) &= \sup_{h \in SE} (f(x-h)) \\ &= \{f(y) | f(y) = \sup(f(z)), z \in SE_x\} \end{aligned} \quad (2.32)$$

and

$$\begin{aligned}\varepsilon_{SE}(f)(x) &= \inf_{h \in SE} (f(x+h)) \\ &= \{f(y) | f(y) = \inf_{z \in \hat{SE}_x} f(z)\}\end{aligned}\quad (2.33)$$

where  $\hat{SE}$  indicates the reflection of the structuring element w.r.t. the origin; e.g.  $\hat{SE} = \{-b | b \in SE\}$ .

These two operators are dual operators with respect to the image complement (e.g.  $f^c(x) = -f(x)$ ):

$$\delta_{SE}(f) = (\varepsilon_{\hat{SE}}(f^c))^c \quad (2.34)$$

Dilation and erosion are increasing operators. This means that if  $f(x) \leq g(x), \forall x \in E$ , then  $\delta_{SE}(f) \leq \delta_{SE}(g)$  and  $\varepsilon_{SE}(f) \leq \varepsilon_{SE}(g), \forall x \in E$ .

Dilation (erosion) is extensive (anti-extensive); e.g.  $f \leq \delta_{SE}(f)$  ( $\varepsilon_{SE}(f) \leq f$ ),  $\forall x \in E$ .

The key element that makes such basic operators the foundation of evolved morphological filtering operators is the duality in the adjunction sense [Serra, 1982; Heijmans, 1995; Najman and Talbot, 2013]

$$\delta_{SE}(g) \leq f \iff g \leq \varepsilon_{SE}(f) \quad (2.35)$$

for every pair of images  $f$  and  $g$ .

Also the two following properties hold for dilation and erosion.

- Distributivity:

$$\begin{aligned}\delta_{SE}(f \vee g)(x) &= \delta_{SE}(g)(x) \vee \delta_{SE}(f)(x) \\ \varepsilon_{SE}(f \wedge g)(x) &= \varepsilon_{SE}(f)(x) \wedge \varepsilon_{SE}(g)(x)\end{aligned}\quad (2.36)$$

- Associativity:

$$\delta_{SE_1 \oplus SE_2}(\delta_{SE_3}(f))(x) = \delta_{SE_1}(\delta_{SE_2 \oplus SE_3}(f))(x) \quad (2.37)$$

where  $\oplus$  indicates the Minkowski addition.

In Figure 2.8 we report the effect of these operators, with several structuring elements depicted aside, on industrial steel images.



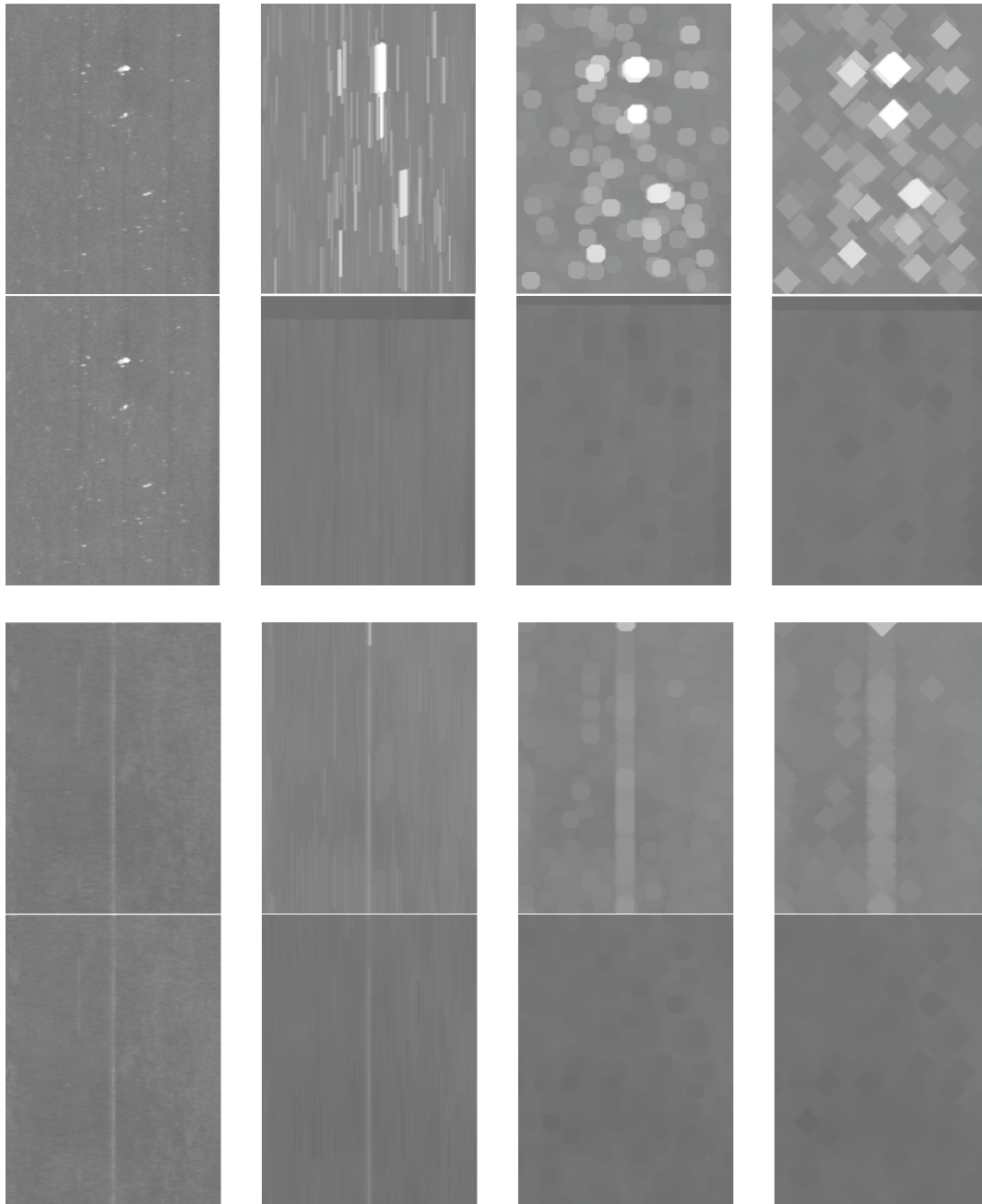


Figure 2.8. Examples of dilation and erosion with three different structuring elements. Leftmost column shows the original image, in this case we have a spot like defect and a vertically elongated one. Then from column 2–4 we have erosion (even rows) and dilation (odd rows) with respectively a vertical line of length 50, a disk of size 10 and a diamond of size 15. It is clear that with the dilation every bright structure that has at least a portion within the SE will be detected (white blobs); with the erosion the structure has to "fit" the SE in order to have a high response.

### Opening and Closing

The composition of the two basic elements of morphological processing, the dilation and erosion, when composed together produce a new set of operators: opening and closing. They are defined as follows

$$\gamma_b(f)(x) = \delta_b(\varepsilon_b(f))(x) \quad (2.38)$$

and

$$\varphi_b(f)(x) = \varepsilon_b(\delta_b(f))(x). \quad (2.39)$$

Also opening and closing are dual operators

$$\gamma_s E(f) = (\varphi_s E(f^c))^c. \quad (2.40)$$

For any two grey-level images  $f$  and  $g$  these operators verify the following properties:

- Increasingness (preserve the ordering):

$$\begin{aligned} f(x) \leq g(x) &\Rightarrow \gamma_{SE}(f)(x) \leq \gamma_{SE}(g)(x) \\ \text{and } \varphi_{SE}(f)(x) &\leq \varphi_{SE}(g)(x). \end{aligned} \quad (2.41)$$

- Idempotence:

$$\gamma_{SE}(\gamma_{SE}(f)) = \gamma_{SE}(f) \text{ and } \varphi_{SE}(\varphi_{SE}(f)) = \varphi_{SE}(f). \quad (2.42)$$

- $\gamma_{SE}(\cdot)$  is anti-extensive:

$$\gamma_{SE}(f)(x) \leq f(x) \quad (2.43)$$

- $\varphi_{SE}(\cdot)$  is extensive:

$$f(x) \leq \varphi_{SE}(f)(x) \quad (2.44)$$

Figure 2.9 shows an example of these operators on the same images used for the dilate and erode case of Figure 2.8. As it can be observed, in comparison with dilation/erosion, which modify always the shape of the objects in the image, opening/closing remove structures which do not fit the structuring element keeping the other object unchanged.

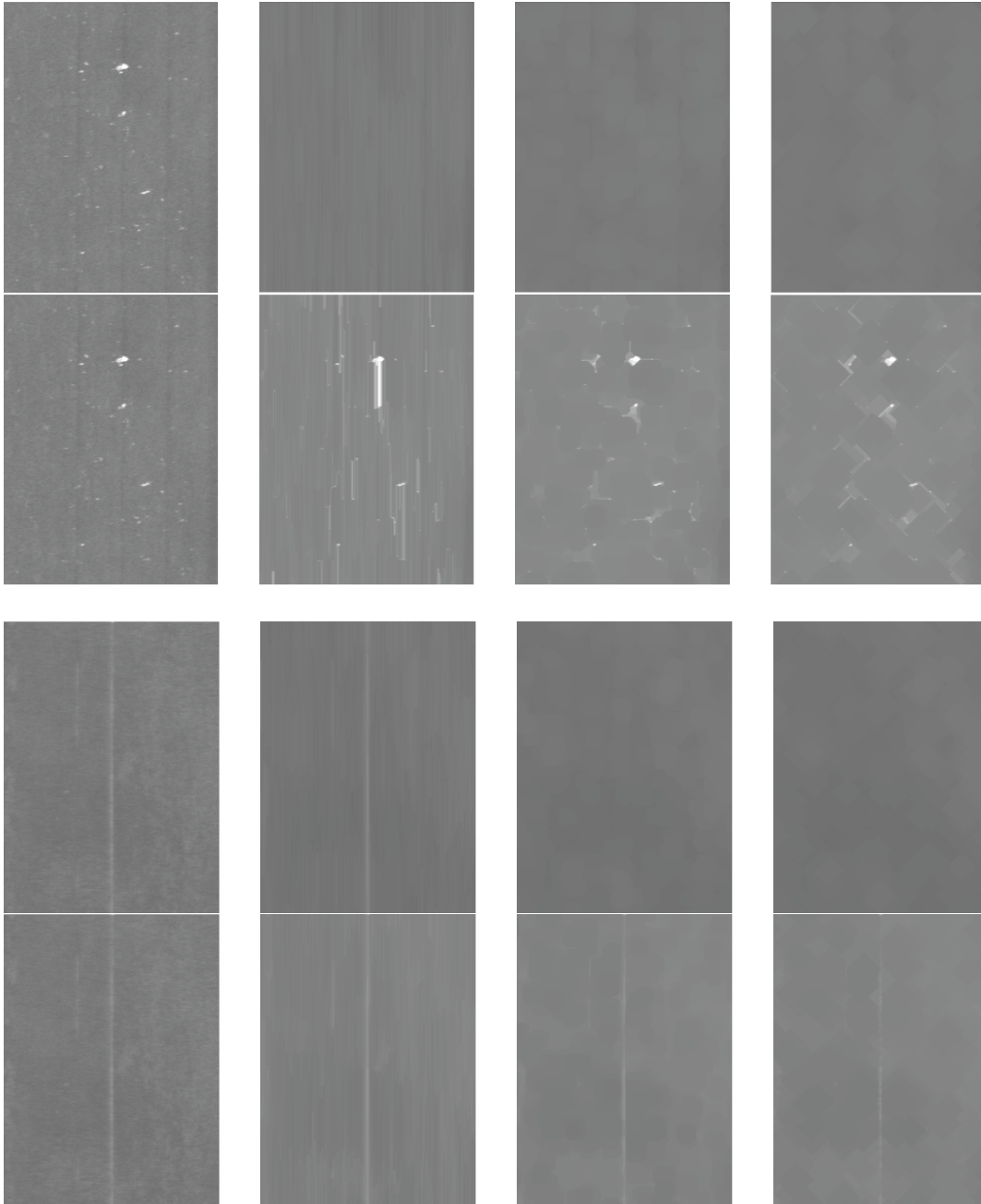


Figure 2.9. Examples of opening and closing with three different structuring elements as in Figure 2.8. Odd rows show the opening and even rows report the closing results. It is interesting to note the behavior on the opening with a vertical structuring element (second column): in the case of the spot defect it is completely removed from the image as the structure is too small to fit entirely the morphological kernel; for the vertically elongated defect instead we have a response on the actual defect and remove partially the noisy background.

### Geodesic reconstruction

Morphological reconstruction is a transformation operating on two images, one called the marker (starting point of the transformation) and the other called the mask (constraints on the transformation). In case of geodesic dilation (erosion has equivalent formulation) we have the set of iterates defined by

$$\delta^i(f, m) = \delta^1(\delta^{i-1}(f, m)), \quad (2.45)$$

where the unitary dilation is given by

$$\delta^1(f, m) = \delta_{SE}(m) \wedge f, \quad (2.46)$$

where  $SE$  is the unitary discrete ball. The resulting reconstruction by dilation is obtained at the fix point

$$\delta_{SE}^\infty(f, m) = \delta_{SE}^i, \quad (2.47)$$

in other words at idempotence, two successive iterates produce the same output. This convergence is guaranteed for a finite image.

We note that the geometric prior given by the structuring element in standard dilation/erosion is here replaced by the marker image. Hence, the choice of the marker plays an important role, as expected. The choice of opening and closing gives us the set of transformations which fall under the name of opening and closing by reconstruction which we briefly describe here.

**Opening and closing by reconstruction.** Using an opening (closing) as marker  $m$  for the reconstruction we have

$$\gamma_{SE}^{\text{REC}}(f) = \delta_{SE}^\infty(f, \gamma_{SE}(f)). \quad (2.48)$$

The interesting property of this transform is that while  $\gamma_{SE}$  (or  $\varphi_{SE}$ ) modifies the contours of the image the reconstruction  $\gamma_{SE}^{\text{REC}}$  efficiently restores them when the objects have not been completely removed.

In Figure 2.10 we show an example of the opening by reconstruction operator with two different structuring elements to show that the borders are effectively preserved with no blurring or connected components deformation as would inevitably happen with conventional linear image processing techniques.

**Criteria opening/closing.** In general *filters by reconstruction* involve the notion of connectivity, i.e. if  $X$  is connected,  $\gamma_{SE}(X) = \emptyset \iff \gamma_{SE}^{\text{REC}}(X) = X$ , and for functions, the opening by reconstruction is given by

$$\gamma_{SE}^{\text{REC}}(f)(x) = \sup\{h \leq f(x) \mid \gamma_{SE}(\gamma_x^c(X_h(f))) = \emptyset\},$$

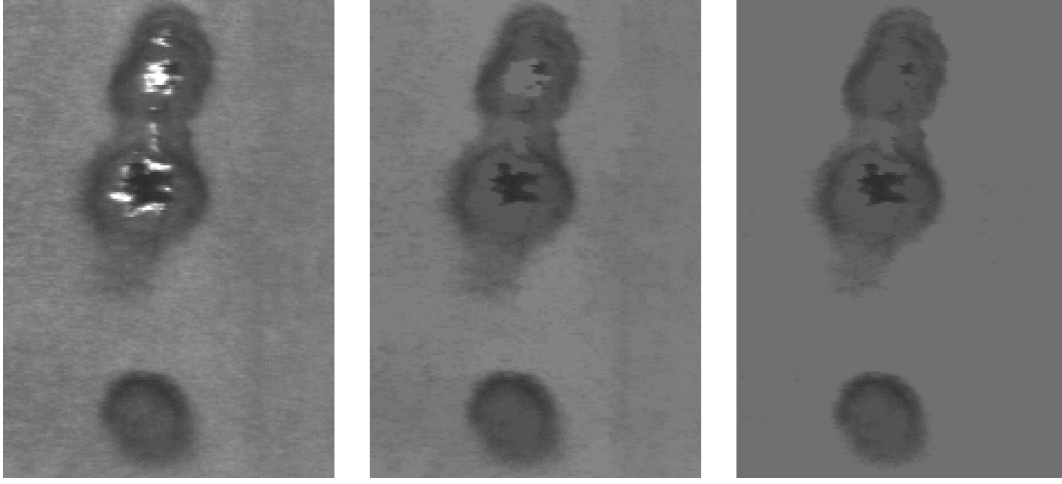


Figure 2.10. Examples of opening by reconstruction using a disk structuring element of size 5 (central) and 50 (right). The original defected sample is shown on the left. Please note how the details are preserved even which such a strong image simplification.

where  $\gamma_x^c(A)$  is the connected component of  $A$  marked by  $x$  (extracts the connected set containing  $x$ ).

**Area opening** is a particular connected operator based on the notion of surface area [Vincent, 1994]. The area opening of  $f$  of parameter  $SE_a$ , denoted by  $\gamma_{SE_a}^a(f)$ , is given by

$$\gamma_{SE_a}^a(f)(x) = \sup\{h \leq f(x) | A(\gamma_x^c(X_h(f))) \geq SE_a\},$$

where  $A(X)$  is the area of  $X$ . The area opening can be seen as the as an opening with a structuring element which locally adapts its shape to the image structures.

### Residue operators

These are a class of operators which are defined as difference of two or more morphological transforms. The simplest is the *morphological gradient*, which is defined in its symmetrical version as

$$|\nabla_{SE}(f)| = \delta_{SE}(f) - \varepsilon_{SE}(f). \quad (2.49)$$

It can also be defined by the dilation residue as

$$|\nabla_{SE}^\delta(f)| = \delta_{SE}(f) - f, \quad (2.50)$$

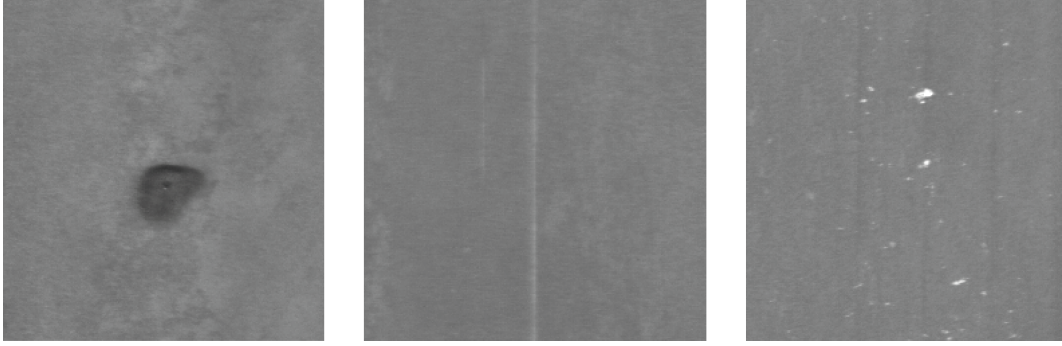


Figure 2.11. The three classes of defects for which a morphological detector based on residual operators needs to be designed.

extracting the external edges of bright structures, or by its dual with the erosion

$$|\nabla_{SE}^{\varepsilon}(f)| = f - \varepsilon_{SE}(f) \quad (2.51)$$

which yields the internal edges of bright objects.

A more interesting operator for the industrial inspection system is the top-hat, which correspond to the difference between the original image and the opening (closing) transform. Consequently, the top-hat produces the bright (dark) structures removed by the opening (closing). We have two varieties of this operator, the *white*

$$\rho_{SE}^{+}(f) = f - \gamma_{SE}(f) \quad (2.52)$$

and the *black*

$$\rho_{SE}^{-}(f) = \varphi_{SE}(f) - f. \quad (2.53)$$

A more detailed description is given in the next section.

### 2.3.2 Morphology in the steel industry

Real industrial applications from steel industry widely use residual operators to detect defects. Let us consider a simplified setting where we have to work with 3 defects with circular, vertical and spot-like structure as illustrated in Figure 2.11. The aim is to design a detector which is able, given an image, to preserve only the defected area putting the remaining part of the image to zero. This way detection comes as cheap as it can be, a simple sum followed by a thresholding over the filtered image pixels.

In Figure 2.12–(a) we show the result of a black top-hat using closing by reconstruction with hexagonal structuring element of size 30 as marker. We

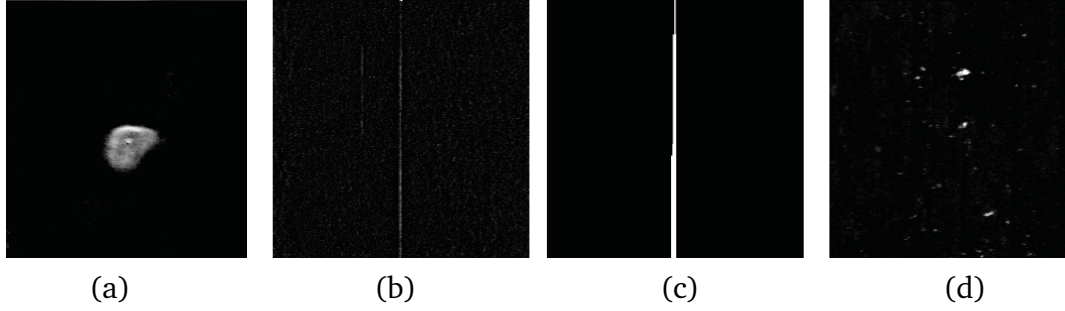


Figure 2.12. Detection results with residual operators on the images shown in Figure 2.11

also perform H-Maxima to improve the visualization. Figure 2.12–(b) shows instead the result of the detector for the elongated defect class where we use a white top-hat (the structure is brighter than the background in this case) with criteria opening with horizontal line of size 5. Then we use a criteria opening with a vertical line of 100 pixels to finally extract the vertical structure shown in Figure 2.12–(c). For the last class of defects considered for this toy example on the application of residual operators to steel industry we have small bright blobs on a darker background.

Of course it would be desirable to have each detector to have a high response for its defect and weak response for the others. With the filtering pipeline designed so far, however, we face a problem with the elongated defect which is detected by the blob structure detector. This is because the opening by reconstruction preserves all the structure which are smaller than the structuring element and in the case of a line all its sub-segments will be preserved, therefore resulting in the entire line being preserved by the transform. In order to fix this issue we apply a morphological criteria opening for the blob detector to keep only objects smaller than  $20px$ . The final result is shown in Figure 2.12–(d).

**Learning challenges** Following the general design ideas of the previous example it does not take long to see the issue of such an approach for designing defect detectors. Finding a correct set of operators, setting the criteria for the markers and the corresponding structuring elements such that the desired behavior is achieved on a complex dataset is a cumbersome and, more importantly, hard to achieve task. Is enough to consider the three defect toy example previously given. When the inter-class variability of each shape is high we immediately start missing defects or to raise false alarms. This, of course, requires additional



human intervention and costs money, in particular when important defects are not detected because of these design choices.

The challenge is to leverage this hand designing process in complex scenarios. We have seen that convnets are trying to achieve a similar objective learning the proper parameterization from a training set. Unfortunately their structure is not suited for morphological operators.

In our study we investigate and propose a formulation of such network tailored for learning of such highly non-linear operators through gradient descent, partially overcoming some of the main limitation of this filter design procedure.

## 2.4 Similarity based metric learning

Similarity is a fundamental notion underlying a variety of computer vision, pattern recognition, and machine learning tasks ranging from retrieval, ranking, classification, and clustering to object detection, tracking, and registration. In all these problems, one has to quantify the degree of similarity between objects, usually represented as feature vectors. While in some cases domain-specific knowledge dictates a natural similarity function, most generally a “natural” measure of similarity is rather elusive and cannot be constructed without side information provided e.g. through human annotation.

The idea of constructing similarity measures suitable to specific data has been thoroughly explored by the statistics and machine learning communities. Methods can be roughly divided into unsupervised and supervised. The former class uses only the data with no additional side information. Unsupervised methods include PCA and its kernelized version [Schölkopf et al., 1997] that approximate the data globally by their second-order statistics either in the original Euclidean space or in a feature space induced by a kernel; and various local embedding methods such as the locally linear embedding of Roweis and Saul [2000], the Laplacian eigenmaps of Belkin and Niyogi [2003], and diffusion maps of Coifman and Lafon [2006], which are all based on the assumption that the data, residing in a high-dimensional Euclidean space, actually belong to a low-dimensional manifold, a parametrization of which is looked for. Unsupervised methods are inherently limited due to their inability to incorporate side information into the learning process.

Supervised methods can be further subdivided according to the type of side information they rely on. Class labels is the most straightforward way of specifying side information, and is used in methods dating back to linear discriminant analysis (LDA) [Johnson and Wichern, 2002] and its kernelized version [Mika



et al., 1999], as well as the more modern approaches of Xing et al. [2002] and Weinberger and Saul [2009].

Other methods accept side information in the form of similar and dissimilar pairs (Davis et al. [2007]) or triplets of the form “ $x$  is more similar to  $y$  than  $z$ ” (Shen et al. [2009]; McFee and Lanckriet [2009]). A family of methods referred to as multidimensional scaling (MDS) rely on metric dissimilarity values supplied on a training set of pairs of data vectors, and seek a Euclidean representation that reproduces them as faithfully as possible (Borg and Groenen [2005]).

More recently, there has been an increased interest in similarity learning methods based on embedding the data in spaces of binary codes with e.g. the Hamming metric [Gong et al., 2012; Gong and Lazebnik, 2011; Kulis and Durrell, 2009; Liu et al., 2012; Norouzi and Blei, 2011; Norouzi et al., 2012; Salakhutdinov and Hinton, 2009; Wang et al., 2010b]. Such an embedding can be considered as a hashing function acting on the data trying to preserve some underlying similarity.

Notable examples of the unsupervised setting of this problem include locality sensitive hashing (LSH) [Andoni and Indyk, 2006; Gionis et al., 1999] and spectral-type hashing [Liu et al., 2011; Weiss et al., 2008], which try to approximate some trusted standard similarity such as the Jaccard index or the cosine distance.

Shakhnarovich et al. [2003] proposed to construct optimal LSH-like hashes (referred to as *similarity-sensitive hashing* or SSH) using supervised learning. More efficient approaches have been subsequently proposed by Torralba et al. [2008b] and Strecha et al. [2012] with impressive performance in large-scale context-based retrieval and image-based localization applications.

In the same setting, a simple method based on eigendecomposition of covariance matrices of positive and negative samples was proposed by Strecha et al. [2012]. In our previous work [Masci et al., 2011b] we posed the problem as a neural network learning allowing for nonlinear embeddings and large scale learning through stochastic gradient optimization.

Hashing methods have been used successfully in various vision applications such as large-scale retrieval [Torralba et al., 2008b], feature descriptor learning [Strecha et al., 2012; Masci et al., 2011b], image matching [Korman and Avidan, 2011] and alignment [Bronstein et al., 2010].

In what follows we report two very well known tools for dimensionality reduction and supervised metric learning. In particular the siamese network is another key model used for our thesis work and for which we propose a novel application and extensions to multi-modal cases.

### Neighborhood Component Analysis

Neighborhood Component Analysis (NCA), is a supervised technique that learns a Mahalanobis distance measure to be used for KNN classification which has been proposed by [Roweis et al. \[2004\]](#). It directly maximizes a stochastic variant of the Leave One Out (LOO) KNN score and can also be used to learn low-dimensional linear embeddings which are useful for visualization and that can help classification as well. Contrary to all feature learning methods presented so far this one is non-parametric. We report it because of its relevance for our purpose and because of its widespread popularity.

Given a dataset of points  $x_0, x_1, \dots, x_n$  in  $\mathbb{R}^n$  and corresponding class labels  $c_0, c_1, \dots, c_n$ , the authors search for a distance metric which maximizes the KNN performance. The authors estimate the Mahalanobis metrics by learning a linear mapping of the input space,  $\mathbf{A}$ , so that  $\mathbf{Q} = \mathbf{A}^T \mathbf{A}$  and  $d(x, y) = (x - y)^T \mathbf{Q} (x - y) = (\mathbf{A}x - \mathbf{A}y)^T (\mathbf{A}x - \mathbf{A}y)$ . As the LOO classification loss is very discontinuous a soft neighbor assignment in the transformed space is adopted. Each point  $p_i$  selects its neighbor  $j$  with some probability  $p_{i,j}$  with the following rule

$$p_{i,j} = \frac{\exp^{-\|\mathbf{A}x_i - \mathbf{A}x_j\|^2}}{\sum_{k \neq i} \exp^{-\|\mathbf{A}x_i - \mathbf{A}x_k\|^2}}, \quad p_{i,i} = 0.$$

Following the previous assignment rule the probability  $p_i$  that a point  $i$  will be correctly classified is given by

$$p_i = \sum_{j \in C_i} p_{i,j}$$

where  $C_i = \{j | c_j = c_i\}$  denotes the points with the same class in the training set.

The final objective, to be maximized, is given by the following formula

$$f(\mathbf{A}) = \sum_i \sum_{j \in C_i} p_{i,j} = \sum_i p_i \quad (2.54)$$

which the authors optimize with gradient based learning.

An interesting view on this optimization is that, maximizing eq. 2.54, is equal to minimize the  $\ell_1$  norm between the true class distribution (having probability one on the true class) and the stochastic class distribution induced by  $p_{i,j}$  via  $\mathbf{A}$ . A natural alternative distance is the KL-divergence which results in the following optimization

$$g(\mathbf{A}) = \sum_i \log\left(\sum_{j \in C_i} p_{i,j}\right) = \sum_i \log(p_i).$$

### Siamese Architecture

The siamese architecture, originally developed by [Bromley et al. \[1993\]](#) and later revisited with the name of DrLim by [Hadsell et al. \[2006\]](#), is a model composed by two copies of the same neural network sharing the same set of parameters; the name follows from this structure. It receives tuples as input, instead of single patterns, and minimizes a loss function which couples the output of the two nets to embed the input samples in nearby locations (e.g. similar pairs collapse to the same point) or at a given distance margin (e.g. dissimilar pairs). A commonly used loss is the following:

$$L(\theta; x_i, x_j, t) = t \frac{1}{2} \|\xi(x_i) - \xi(x_j)\|^2 + (1-t) \frac{1}{2} (\max\{0, m - \|\xi(x_i) - \xi(x_j)\|\})^2, \quad (2.55)$$

where the constant  $m$  represents the margin (e.g. hinge loss) between dissimilar pairs. The margin is introduced as regularization to keep the system from minimizing the loss just pulling two vector as far apart as possible.  $t \in [0, 1]$  is the similarity measure, 1 for similar pairs and 0 for dissimilar pairs.

The gradient of the model is computed as follows

$$\frac{\partial L}{\partial \theta} = \begin{cases} (\xi(x_i) - \xi(x_j)) \frac{\partial \|\xi(x_i) - \xi(x_j)\|}{\partial \theta} & \text{if } t = 1 \\ (-m - (\xi(x_i) - \xi(x_j))) \frac{\partial \|\xi(x_i) - \xi(x_j)\|}{\partial \theta} & \text{if } t = 0 \text{ and } \|\xi(x_i) - \xi(x_j)\| < m \\ 0 & \text{otherwise} \end{cases}$$

The optimization is performed with stochastic updates as exemplified in Algorithm 2. Tuples are randomly sampled from the training set and, according to some known similarity a similar/dissimilar labels is generated. The model is updated with the current mini-batch of samples and the process iterates until convergence is reached.

Networks of this type can also be traced back to the work of [Schmidhuber and Prelinger \[1993\]](#) on problems of predictable classification. Because of this structure it is well suited to solve metric learning problems; [Hadsell et al. \[2006\]](#) used it to learn an invariant mapping of tiny images directly from pixel representation and [Taylor et al. \[2011\]](#) to learn a model that is highly effective at matching people in similar pose which exhibits invariance to identity, clothing, background, lighting, shift and scale.

In [Masci et al. \[2011b\]](#) and [Masci et al. \[2014b\]](#) we introduced a novel approach for the similarity sensitive hashing problem to learn an invariant mapping for omnidirectional feature matching. In this application the lens distortion

---

**Algorithm 2:** Online training algorithm for the siamese network.

---

**Data:**  $\mathbf{X}, \mathbf{T}$ , model  $nn$

**repeat**

    Sample tuple  $x_i, x_j$

    Obtain ground-truth similarity  $t$

$\begin{cases} \text{update } \theta \text{ to decrease } \|\xi(x_i) - \xi(x_j)\| & \text{if } t = 1 \\ \text{update } \theta \text{ to increase } \|\xi(x_i) - \xi(x_j)\| & \text{if } t = 0 \end{cases}$

**until** convergence;

---

foils most of the *invariances by construction* of descriptors such as SIFT (not designed for the task) and shows superior performance w.r.t. previous state-of-the-art methods. The aim of the work was to validate the hypothesis that hash functions could be learned using siamese-like architectures.

For retrieval a similar system, which aims in optimizing directly the precision@k is proposed in [Weston et al. \[2010\]](#) whereas a former application can be found in [Chechik et al. \[2010\]](#). In Chapter 8 we extend the conventional framework to deal with multimodal data via a cross-modality coupling loss. Hash codes for the similarity sensitive hashing problem of our method achieve state-of-the-art performance on challenging tasks and shows the ability to learn complex mapping by stacking multiple layers of non linearity.

## Chapter 3

# Classifying defects with MPCNN

MPCNN have achieved state-of-the-art performance in a variety of pattern recognition problems, but have never been applied to steel defect recognition. This chapter presents such an approach for supervised steel defect classification. This is the foundation work on the subject for my thesis and its aim is to validate the viability of convnets as classifier for steel industry and to motivate further investigations.

After a brief discussion on the most common features used in steel industry and computer vision we proceed to a thorough comparison of such fixed feature extractors to convnets. First, each of the classical features is used alone with both softmax and SVM with Gaussian kernel classifiers to assess individual performance when applied to steel. Then, as features usually tend to complement each other, an ensemble of classifiers is also build to strengthen performance. In both cases we show that a fairly low complexity convolutional net trained on the same data achieves an error rate of 7%, at least two times better than the best conventional competitor or its ensemble. Not only better results are obtained, but the proposed method also works directly on raw pixel intensities of detected and segmented steel defects, avoiding further time consuming and hard to optimize ad-hoc preprocessing.<sup>1</sup>

### 3.1 Standard Features

A standard inspection system such as the one described in Chapter 2 relies heavily on hand-crafted feature extraction. The following describes the most widely used algorithms which have achieved state-of-the-art performance for textile and

---

<sup>1</sup>This chapter is based on [Masci et al. \[2012\]](#).

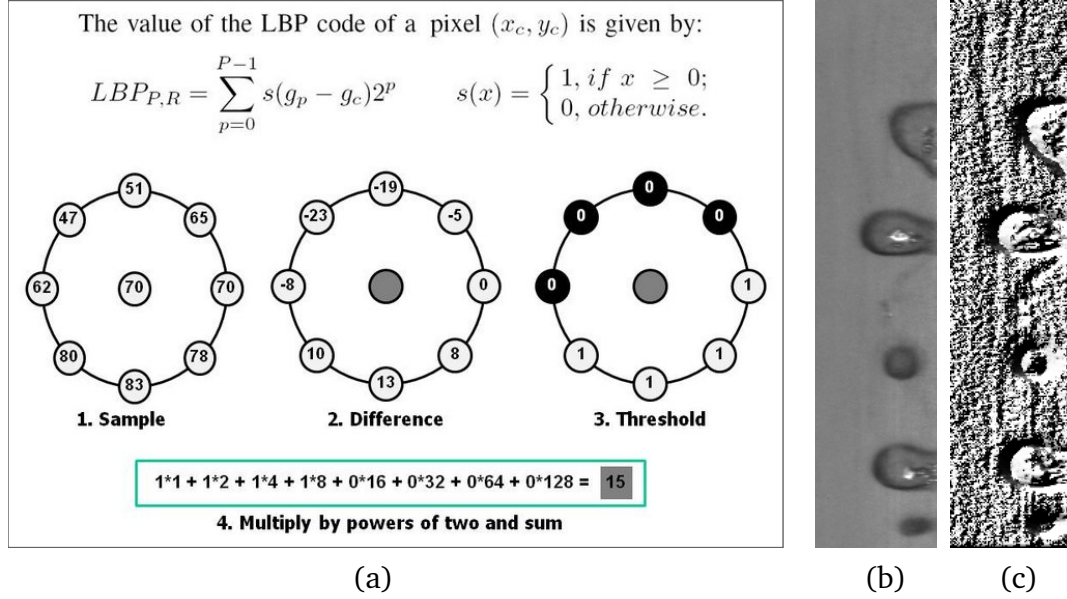


Figure 3.1. (a): A schematic representation of how a LBP descriptor is computed; (c) exemplar LBP image when applied to the steel defect image (b).

steel inspection systems.

- *Local Binary Patterns (LBP)* [Ojala et al., 1996] is an operator that focuses on the spatial structure of grey-level texture. Figure 3.1 illustrates how LBP is computed. For each pixel, a binary descriptor is generated, which is intensity and rotation invariant, based on the sign of differences between neighboring pixels. After this is done for every pixel in the image, a histogram can be computed or the new representation can be used in form of an “image”.
- *Local Binary Pattern Histogram Fourier (LBP-HF)* [Ahonen et al., 2009] is a rotation invariant descriptor computed from the discrete Fourier transform of LBP. The rotation invariance is computed on the histogram of non-invariant LBP, hence the rotation invariance is attained globally and not locally. The resulting features are invariant to rotations of the whole input signal but still retain information about the relative distribution of different orientations of uniform local binary patterns.
- *Monogenic-LBP* [Zhang et al., 2010] integrates the traditional LBP descriptor with two rotation invariant measures, the local phase and the local surface computed by the 1st- and 2nd-order Riesz transforms.

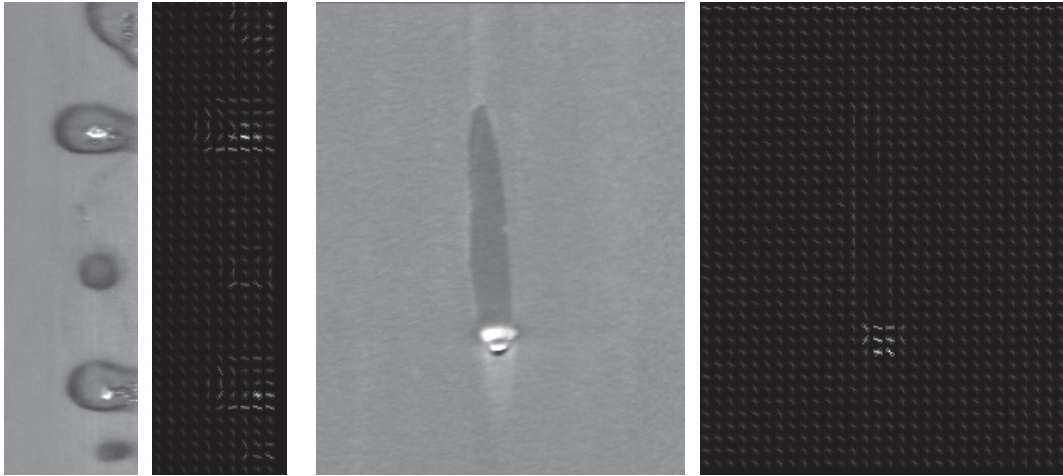


Figure 3.2. Visualization of HOG descriptors on industrial steel defect images. The detector is able to capture interesting points in the image and describes them with their gradient orientation. In cases where the background is highly textured, however, such an approach tends to fail.

- *Rotation invariant measure of local variance (VAR)* [Ojala et al., 2002] is a rotation invariant descriptor which incorporates information about the contrast of local image texture. Returns a histogram of the variance, calculated on a circumference of a given radius and quantized into equally spaced sample points. In conjunction with LBP makes a very powerful rotation invariant descriptor of local image texture.
- *Histogram of Oriented Gradients (HOG)* [Dalal and Triggs, 2005] is based on evaluating normalized local histograms of image gradient orientations over a dense sampling grid. The basic idea is that local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions. Figure 3.2 demonstrates this descriptor when applied to steel defect images.
- *Pyramid of Histograms of Orientation Gradients (PHOG)* [Bosch et al., 2007] is an extension of the HOG descriptor that also considers the spatial locality of the descriptor's constituents. Its construction follows the image pyramid decomposition as proposed by Lazebnik et al. [2006].



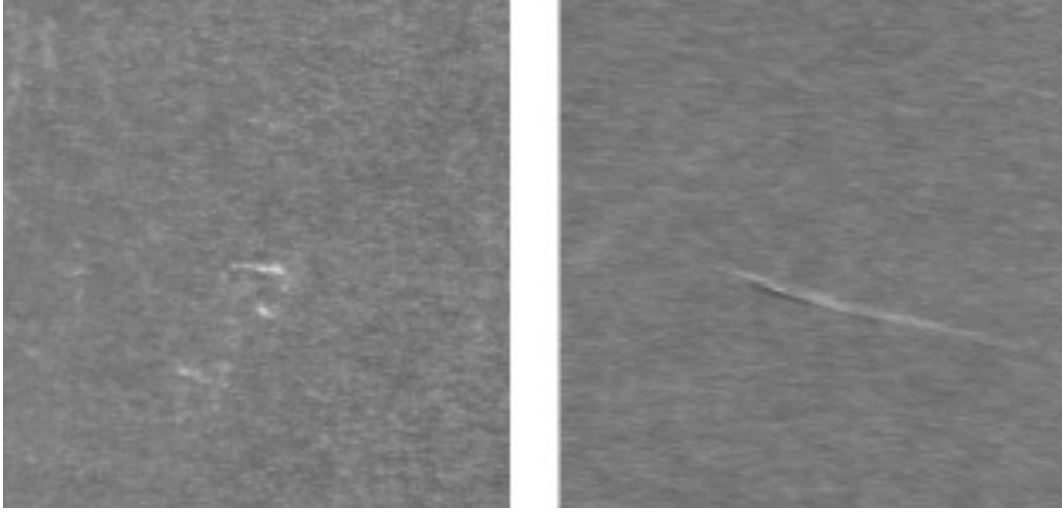


Figure 3.3. Two instances of the same defect class as labeled in our dataset.

## 3.2 Dataset

The MPCNN was tested on data collected from an actual production line of our industrial partner ArcelorMittal. The dataset is composed of 30 defect classes with images spanning from less than 100px to more than 1000px in each dimension. A subset of 7 classes with images of similar size and sufficiently hard to discriminate were selected to assess whether the MPCNN can improve over the state-of-the-art. In Figure 3.3 two instances of the same defect are shown to illustrate how much defects can vary within a class. The images come out of the detection phase and therefore are big patches extracted from the original steel coil images. Each image has an annotation that indicates the region containing the defect (ROI); this way we have some additional background information at our disposal. In total the training set consists of 2281 images and the testing set consists of 646 images.

The detection stage can obviously miss the defect and create false alarms, therefore the dataset may contain background patches labeled as defected pieces of steel. This means that, in addition to high intraclass variability, the classification algorithm also has to deal with possible false positives in the training set.

In contrast with classical machine vision approaches, where there are no constraints on the input dimensions in the feature extraction stage, MPCNN require a fixed size input as they perform feature extraction and classification



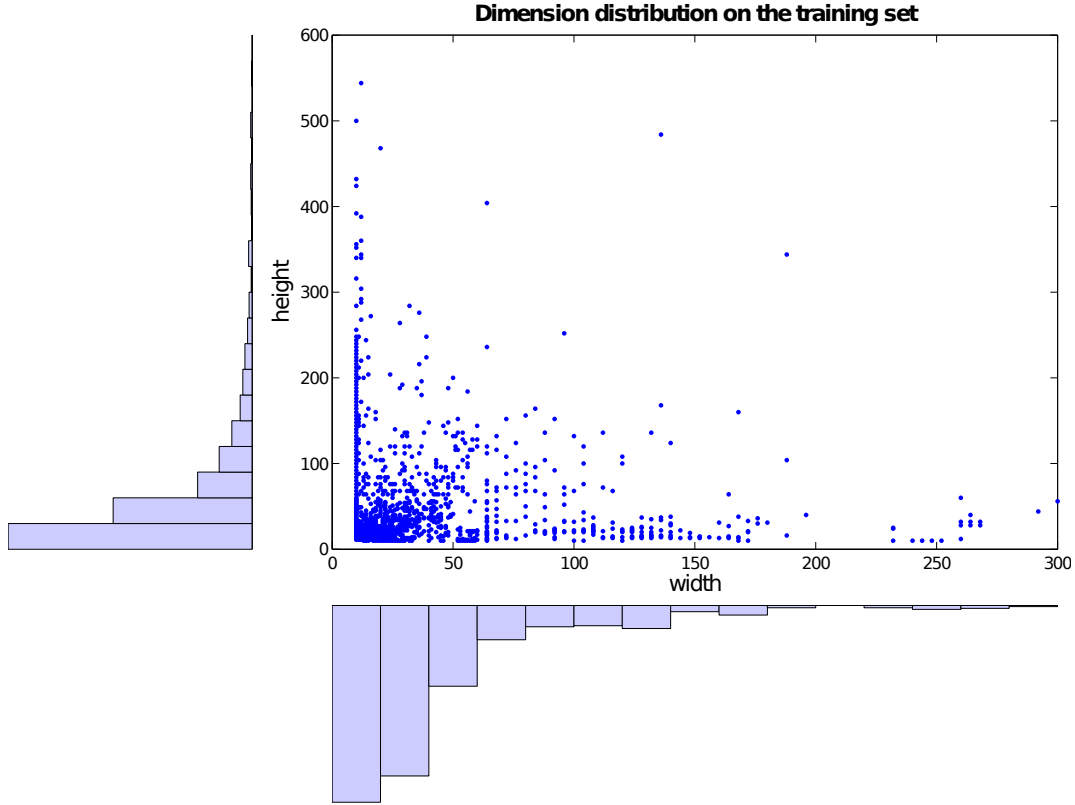


Figure 3.4. Each point denotes the width and height of an image from the training set, histograms of the width and height distribution are also shown. It can be seen that most of the images are smaller than 150px in width and 200px in height. This distribution is used to empirically select the nominal size of the input images for the MPCNN.

jointly; e.g. the feature layer size is function of the input image size. Therefore, the defects are resized, preserving their aspect ratio and minimizing the overall down/up-sampling rate according to the distribution of image dimension over the training set, which is reported in Figure 3.4.

The images were resized to  $150 \times 150$  pixels: images larger than these dimensions were downsampled, smaller images were padded with pixels around their borders. In Figure 3.5 a sample from each of the classes on the superior (first row) and inferior (second row) part of the steel strip is shown. Unfortunately, there is no correspondence between images from the two parts. This is a good example to show how hard is to model a feature extraction which is invariant to such variability. We purposely merged the top and bottom parts into

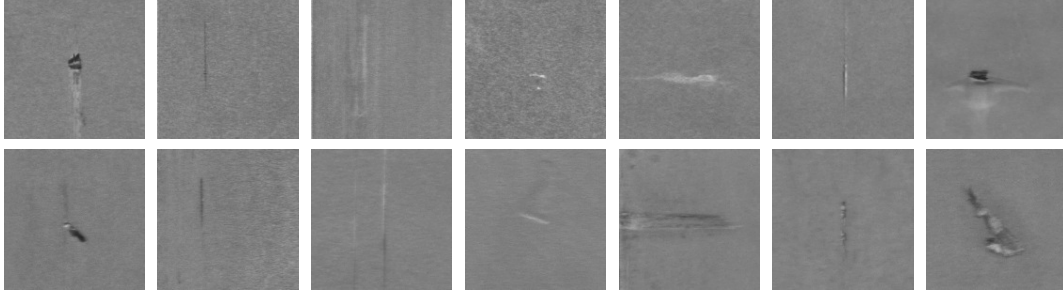


Figure 3.5. A sample from each of the seven defects in the dataset. First row: superior part of the strip; second row: inferior part of the strip. There is no correspondence between the two images for a given instance of the defect.

the same class to make the task more challenging and also to avoid having very few images for a single class.

In classical approaches, a histogram of the features is created in order to obtain a constant sized feature vector used for classification. For all experiments, when using classical feature extraction techniques, we adopt the following approach. The original images are used without any resizing to avoid artifacts that might ruin the quality of the produced features. For example, if a defect covered only 10% of the image and we zero-padded, the resulting histogram would almost be flat and the actual information regarding the defect would be lost.

### 3.3 Experimental Setup

For each of the standard features we tested, the code and the configurations suggested by the respective authors was used, so some margin of improvement might be achieved by fine tuning the parameters. For LBP we combine rotation invariant and non rotation invariant features by concatenating the two feature vectors which we empirically found to deliver better performance. As always, prior knowledge and experience are important, but fine tuning the parameters of the feature extraction is usually harder than for MPCNN, especially for LBP which depends on many parameters and comes in many variants.

The extracted features, which are histograms, are locally normalized between 0 and 1 and then fed to a classifier. We opt for a multi-layer perceptron (MLP) just as for the MPCNN, where the last two layers are MLPs. It is also easier to extract values that can afterwards be interpreted as posterior probabilities (see section 3.4.3). We used a single layer architecture with 100 hid-

den nodes<sup>2</sup>. The number of inputs is given by the dimensionality of a given feature representation and the *tanh* activation function is used for the hidden layer; training is performed for a total of 300 epochs. The output layer has 7 neurons, one for each defect class, that are normalized with a softmax activation function. We also tested a SVM [Chang and Lin, 2011] with RBF-kernel, whose kernel parameter  $\gamma$  and penalty term  $C$  are optimized over following grid:  $\gamma = [2^{-15}, 2^{-13}, \dots, 2^3]$  and  $C = [2^{-5}, 2^{-3}, \dots, 2^{15}]$  by a 5-fold cross-validation.

Combining the output of several classifiers is an easy and effective way of boosting the performance. If the errors of different classifiers have zero mean and are uncorrelated with each other, then the average error might be reduced by a factor of  $M$  simply by averaging the output of the  $M$  models [Bishop, 2006]. In practice, errors of models trained on similar data tend to be highly correlated. To avoid this problem predictions of various classifiers trained on differently normalized data can be combined [Meier et al., 2011]. Along similar lines the same classifier can be trained on random subsets of the training set (bootstrap aggregation technique [Breiman, 1996]), or different types of classifiers can be trained on the same data [Grosicki and El-Abed, 2011]. Here we combine classifiers trained on different features, harnessing the complementary information content of the various image descriptors creating an ensemble of classifiers which is much more robust than each of the single ones. This is the hardest challenge for the MPCNN, a single model which has to be compared with a collection of powerful classifiers.

Two different MPCNN architectures were trained using stochastic gradient descent and annealed learning rate. As data scarcity is severe we perform experiments with and without random translation of max.  $\pm 15\%$  of the image size to augment the available training data to prevent overfitting. Since no padding region is used, however, border effects arise. In order to minimize these we opt for the simplest solution and assign to each unknown pixel the intensity of its closest pixel with respect to the Euclidean distance. This may introduce problems when a defect is close to the border as it might possibly disappear due to the translation factor. Training stops when either the validation error becomes 0, the learning rate reaches its predefined minimum or there is no improvement on the validation set for 50 consecutive epochs. The undistorted, original training set is used as validation set. A GPU implementation of a MPCNN [Ciresan et al., 2011b] is used to speed-up training; on a GeForce GTX 460 one training epoch takes about 90s, a speed-up of 20-40 (depending on the network topology) with

---

<sup>2</sup>We also tested MLPs with more hidden units and an additional hidden layer with no considerable improvement.

Table 3.1. Detailed networks structure. The time per sample refers to the time required for a trained network to produce the class prediction.

Network	#parameters	#connections	time per sample
5HL-MPCNN	1.35M	688M	11.3ms
7HL-MPCNN	622k	295M	6.2ms

respect to an optimized single-core CPU implementation.

The first architecture has 5 hidden layers, a convolutional layer with 50 maps and filters of size  $19 \times 19$ , a max-pooling layer of size  $4 \times 4$ , a convolutional layer with 100 maps and filters of size  $13 \times 13$ , a max-pooling layer of size  $3 \times 3$ , a fully connected layer with 100 neurons, a fully connected layer with 7 output classes (5HL-MPCNN). The second architecture has 7 hidden layers, a convolutional layer with 50 maps and filters of size  $11 \times 11$ , a max-pooling layer of size  $4 \times 4$ , a convolutional layer with 100 maps and filters of size  $6 \times 6$ , a max-pooling layer of size  $3 \times 3$ , a convolutional layer with 150 maps and filters of size  $5 \times 5$ , a max-pooling layer of size  $3 \times 3$ , a fully connected layer with 100 neurons, a fully connected layer with 7 output classes (7HL-MPCNN).

## 3.4 Results

### 3.4.1 Standard Features

Results for SVM and MLP classifiers trained on image descriptors generated by classical features are shown in Table 3.2-top. We see that both classifiers perform nicely with SVM being more robust and less prone to overfit on such small amount of data. The best descriptor, PHOG, achieves an error rate of 15.48.

### 3.4.2 MPCNN

Results are reported for fully trained models, where all layers are trained, and also for models where the first convolutional layer is kept fixed during learning, indicated by RC in Table 3.2. That is, the first convolutional layer performs random filtering, reducing the number of free parameters and hence training time without degrading classification performance. As a matter of fact it even improves generalization when no translations are used [Saxe et al., 2011]. Each

Table 3.2. Classification results for the several methods on the steel classification dataset. Classical features show results as SVM/MLP. For MPCNN the best run is presented along with, in parenthesis, the mean performance and standard deviation among 5 different runs. RC indicates that the first convolutional layer performs a random projection, not trained.

Feature (#dims)		Test Error Rate %
LBP (274)		26.79 / 32.38
LBP-HF (76)		29.11 / 40.09
MONO-LBP (540)		19.66 / 21.20
VAR (10000)		42.88 / 43.34
HOG (81)		19.35 / 21.36
PHOG (680)		15.48 / 19.04
5HL-MPCNN	No Trans.	13.78 (14.83 $\pm$ 0.8)
	Trans.	<b>9.60</b> (9.81 $\pm$ 1.2)
7HL-MPCNN	No Trans.	10.99 (12.38 $\pm$ 1.2)
	Trans.	<b>6.81</b> (8.11 $\pm$ 1.1)
5HL-MPCNN (RC)	No Trans.	12.07 (13.59 $\pm$ 1.1)
	Trans.	<b>10.99</b> (11.98 $\pm$ 0.8)
7HL-MPCNN (RC)	No Trans.	8.20 (9.00 $\pm$ 0.6)
	Trans.	<b>6.97</b> (7.56 $\pm$ 0.5 )
HOG, PHOG, LBP-HF		11.45
LBP, HOG, PHOG		13.46
PHOG, MONO-LBP		10.99
ALL FEATURES		11.60

experiment is repeated five times with different random initializations, since any iterative gradient based optimization technique depends on the starting model. The deeper net yields better results and translating images prior to training by a maximal amount of 15% further increases performance.

All the MPCNN yield lower error rates than any of the feature based classifiers. The best 7HL-MPCNN, with an error rate of 6.97% with and 8.20% without translations, clearly outperforms the best feature based classifier, PHOG, with an error of 15.48%. This illustrates the power and potential of the proposed architecture for defect classification in textured materials.

In Figure 3.6-left the confusion matrix of the best MPCNN from Table 3.2 is

	0	1	2	3	4	5	6
0	0.91	0.01	0.01				0.07
1	0.09	0.85					0.07
2	0.01		0.99				
3	0.04			0.94	0.02		
4	0.02			0.02	0.97		
5		0.05	0.11			0.84	
6	0.14						0.86

	0	1	2	3	4	5	6
0	0.80	0.01	0.01	0.08	0.02	0.01	0.08
1	0.11	0.89					
2	0.03	0.02	0.94			0.01	
3	0.14		0.02	0.83	0.01		
4				0.09	0.91		
5	0.05	0.19				0.76	
6	0.25				0.01		0.74

	0	1	2	3	4	5	6
0	0.85	0.01		0.03	0.01	0.01	0.11
1	0.12	0.88					
2	0.01		0.98			0.01	
3	0.09		0.01	0.88	0.02	0.01	
4			0.02		0.98		
5	0.05	0.05		0.14		0.76	
6	0.20						0.80

Figure 3.6. Confusion matrices for the best classifiers. Left: MPCNN, middle: PHOG, right: PHOG + MONO-LBP committee. Only on defect number two the classical features obtained a better result than that of the MPCNN. Also note the non marginal improvement of a committee w.r.t. the single best classifier.

shown, where the rows represent the true classes and the columns the predicted classes. On the diagonal the per class percentage of correctly classified samples is shown, all off-diagonal entries in each row correspond to the wrongly classified samples for a particular class. For example, 14% (first entry in row 6) of the images from defect class 6 are wrongly classified as class 0.

### 3.4.3 Committee of classifiers

Table 3.2-bottom shows the results of the three best committees out of all possible committees with at least 2 out of the 6 classifiers trained on the 6 different feature descriptors. The best committee decreased the error rate by 5% with respect to the best single classifier. Note, however, that even the three best committees have a much bigger error rate compared to the MPCNN. In Figure 3.6 we clearly see that using a committee of classifiers considerably boosts the recognition rate (compare middle and right matrices). We can also see that a simple MPCNN performs always better in the per-class evaluation (diagonal values) except for defect number 2 where a committee reaches almost perfect accuracy.

## Chapter 4

# General steel defect recognition

In Chapter 3 MPCNN have been successfully applied to the steel defect classification problem. Without prior knowledge excellent results are achieved, outperforming any classifier trained on feature descriptors commonly used for defect classification in textured materials.

However, in order to apply MPCNN a subset of the original dataset had to be created so that images could be resized without altering their content. This is because the classifier feature size of a convnet is function of the input image size and once defined it cannot be changed. General steel defect classification systems have instead to deal with images whose maximum edge size can be as large as  $40\times$  the minimum one (e.g. in the original dataset of Chapter 3 images range from  $\approx 50px$  up to  $\approx 2000px$ ). In these cases, resizing alters the size of the defect, which is a crucial discriminative factor. Each class has usually a seriousness index which is in many cases indeed given by the defect size. Any resizing will be likely to change the class of the defect and therefore will result in poor performance; e.g. severe defects downgraded to acceptable levels, an inadmissible option for ArcelorMittal.

A naïve way of circumventing this problem could be to divide the original training data in subsets of common size and train a MPCNN on each one of those, following the same approach of Chapter 3. Unfortunately, images for every class are spread among the several size clusters and there are degenerate cases where a class or several are not present in all subsets. In this latter situation, combining the various models is very hard and often impossible. Industrial labeled data is expensive, and this requires systems able to learn from few samples per class. Furthermore, the desired invariances for general steel defect recognition are not that easily synthesized to virtually increase the training set. For some defect classes the position is important too and the simple translations used in

Chapter 3 could indeed hinge on performance. A distortion/deformation-free approach is thus preferable.

An additional aspect of MPCNN as defined in Section 2.2.2 is that they lack an explicit way to deal with image features which are visible only at a particular scale. Let us assume, for example, that a particular and important structure in the image can be captured only at the first convolutional layer's output. After a pooling stage such information might be lost forever and, even if in principle it could be possible to learn to propagate it in the deeper layers, it is very hard in practice.

This chapter<sup>1</sup> introduces the Multi-Scale Pyramidal Pooling network (MSPyr-Pool) which aims at solving the general steel defect recognition problem. It is inspired by successful computer vision approaches which do not suffer the resizing problem and consists of three new ingredients:

- a Pyramidal Pooling layer which produces a fixed-dimensional feature vector independent of the input image size;
- a learnable encoding layer to incorporate commonly used encoding strategies of computer vision which aims in limiting overfitting and improving generalization when data is scarce;
- a Multi-Scale feature extraction strategy to compensate for information which may be lost when reaching the classification stage of MPCNN.

Next section introduces the computer vision classifier which inspired this work. Then the new model is introduced and validated on several standard and industrial benchmarks.

## 4.1 Background

In computer vision the state-of-the-art model for image classification extracts local features, such as SIFT descriptors [Lowe, 1999], over a densely sampled grid (e.g. every 16 pixels in each dimension). In practice image patches are extracted and the image is represented with the collection of spatially located low-level features, one for each point in the grid. The next step is to transform this general description of the image to fit the specific task. This is achieved through an encoding stage which quantizes the descriptors into overcomplete and sparse codes commonly known as visual words. The idea is to discover local

---

<sup>1</sup>This chapter is based on Masci et al. [2013c].



object parts which can be used to describe the image via simple histograms; e.g. summing all encoded vectors gives a measure of how often each visual word has been used. This is similar to the bag-of-words representation in natural language processing and because of this it takes the name of bag-of-features (BoF) [Sivic and Zisserman, 2003; Csurka et al., 2004].

To summarize, feature vectors extracted from all or a subset of the training images are clustered. The cluster centers form the set of basis, the visual dictionary, that is used in the encoding stage. Finally a supervised classifier is trained to classify the histogram representation of the image.

The most appealing property of such an approach is that it is applicable to any image size and that, thanks to the feature encoding stage, the discriminative power can be tuned, therefore limiting overfitting. Most of the systems based on BoF work well with only very few training images per class (e.g. 15, 30) contrary to convnets where large datasets are a must.

Unfortunately BoF does not discover new and possibly very discriminative features because it relies on fixed feature extractors. This makes the encoding stage crucial to obtain good performance.

Convnets, contrarily, try to map the pixel-based representation directly into a label vector, learning both feature extraction and encoding from a labelled dataset. Learning the features helps to make the system easily applicable to domains where prior knowledge is not well consolidated and is shown to achieve better recognition performance as reported in Chapter 2.

There are obvious similarities between BoF and fully supervised MPCNN and such similarities inspired and motivated the work presented in this chapter. Both methods extract features based on photometric discontinuities (e.g., edges), either engineered or learned from samples, followed by an encoding stage. Standard MPCNN lack multiple resolution pooling and the explicit encoding steps of winning BoF approaches. In turn, BoF lacks tunable feature extraction stages that may make complex encodings less important. A main drawback of MPCNN is their restriction to constant size input images, which the steel industry cannot simply overcome by resizing and padding.

Inspired by BoF, for its ability to work with very limited number of samples per class and for its input size agnosticism, the rest of the chapter investigates ways of adding these two key ingredients to MPCNN.

In what follows the two main concepts of BoF, namely the feature encoding and pooling, are presented. The former maps each local descriptor into a sparse representation and the latter produces the final image descriptor.

### 4.1.1 Feature encoding algorithms

Features are extracted using engineered approaches (e.g. SIFT or HOG descriptors). What usually varies is not the feature extraction procedure *per-se* but where in the image to extract the descriptors. The most successful methods extract features over a dense, equally spaced grid as in [Bosch et al. \[2007\]](#). Recent improvements in classification performance on commonly used benchmarks [[Wang et al., 2010a](#)] are rather due to improved encoding strategies than new feature descriptors. The quantization step is crucial to produce encodings with the right level of detail that avoid overfitting and lead to improved generalization to unseen data. The de-facto standard for this procedure is given by overcomplete and sparse encodings of the local feature descriptors.

Here three of the most commonly used algorithms for feature encoding are reported. They range from the simplest up to the current state-of-the-art method. It is later shown how to use these approaches in the fully supervised framework of the MSPyrPool network here presented.

1. **Vector Quantization (VQ).** In its original formulation, the spatial pyramid matching uses k-means to obtain a dictionary of centroids (i.e. clusters),  $\mathbf{B} = [b_1, b_2, \dots, b_N]$ , which is then used to solve the following least squares problem

$$\begin{aligned} \underset{\mathbf{c}}{\operatorname{argmin}} \quad & \sum_i^N \|x_i - \mathbf{B}c_i\|^2 \\ \text{s.t. } \forall i. \quad & \|c_i\|_0 = 1, \|c_i\|_1 = 1, c_i \geq 0 \end{aligned} \quad (4.1)$$

where the  $\ell_0$  constraint ensures that only one basis will be on and the  $\ell_1$  constraint ensures that its coefficient value will be 1. In practice a 1 is placed at the position of the nearest neighbor centroid, producing an approximation of  $x_i$  using a single code vector.

2. **Sparse Coding (SC).** Using just a single basis vector results in high quantization errors. Relaxing the  $\ell_0$  constraint in eq. 4.1 allowing more than one active basis reduces the quantization error dramatically. Since the number of bases is bigger than the number of input dimensions the resulting system is underdetermined. An effective way to find a solution is to impose sparsity and reformulate it to the well studied sparse coding problem:

$$\underset{\mathbf{c}}{\operatorname{argmin}} \quad \sum_i^N \|x_i - \mathbf{B}c_i\|^2 + \lambda \|c_i\|_{\ell_1} \quad (4.2)$$

This produces better reconstruction and together with a linear SVM outperforms non-linear approaches on the Caltech-101 benchmark [Yang et al., 2009].

3. **Locality-constrained Linear Coding (LLC).** A more powerful quantization algorithm by Wang et al. [2010a] exploits the locality principle to obtain sparse representations as “locality leads to sparsity but not vice-versa”. LLC solves the following equation:

$$\operatorname{argmin}_{\mathbf{c}} \sum_i^N \|x_i - \mathbf{B}c_i\|^2 + \lambda \|d_i \circ c_i\|^2 \quad (4.3)$$

where  $\circ$  denotes the element-wise multiplication and  $d_i$  represents a regularizer to favor quantization using bases similar to  $x_i$ . LLC is very effective in commonly used benchmarks such as Caltech-101, Caltech-256 and PASCAL VOC and can be computed very efficiently using only the  $k$ -nearest neighbors of the dictionary to reconstruct  $x_i$  instead of explicitly solving eq. 4.3.

#### 4.1.2 Feature pooling

Once the features are encoded, a histogram of active visual words is formed through a procedure called pooling. This is important to put together the local information of low level codes into a fixed representation which describes the entire image. The naïve approach of achieving this, used in early BoF systems, is to sum all the  $N$ -dimensional codes extracted from all the grid points, where  $N$  represents the number of bases in the dictionary, thus producing a global representation. This global descriptor is already very powerful, but information about the spatial relations between image parts is important.

Lazebnik et al. [2006] presented a histogram generation technique where features are considered in their spatial locality. This allows higher level information to be incorporated such as, e.g. “the sky is above the ground” and “the head is on the shoulders”. To obtain such pooling, features are divided using a quad-tree data structure. At every level  $l$  of the tree,  $2^l$  tiles are produced, and for each one a histogram is extracted through pooling from the encoded features belonging to the corresponding tree node. This approach is also used in the PHOG descriptor [Bosch et al., 2007], an improvement over HOG [Dalal and Triggs, 2005]. Several methods to pool have been presented, such as sum-,

average-, and  $\ell_2$ -pooling. Also in this case, as in MPCNN, max-pooling is favorable.

Because such process goes from coarse (e.g. the entire image is pooled) to fine (e.g. tiny portions of the image are pooled) it usually takes the name of pyramidal pooling.

## 4.2 Multi-Scale Pyramidal Pooling Network

In this section the MSPyrPool architecture, which incorporates ideas from BoF to make convnets applicable to general steel defect recognition, is introduced.

Clearly MPCNN and BoF both extract low level features using convolutional filters and pooling operations, but, the filters of a MPCNN are learned from the data, whereas for BoF the feature extractors are fixed. For example, HOG uses the image gradient, which can be computed using convolutional filters and then pools to obtain the final descriptor. SIFT does a similar thing; an input patch of usually  $64 \times 64$  pixels is tiled into 16 quadrants and for each of them a 8-dimensional vector of gradient orientation is extracted. The resulting descriptor is a concatenation of such vectors, resulting in a 128-dimensional feature vector positioned at the centre of the input patch.

Furthermore, BoF is inherently a single layer, whereas deep multilayer architectures are capable of extracting more powerful features [Jarrett et al., 2009].

In what follows we introduce two new layers that compose the MSPyrPool framework.

### 4.2.1 Pyramidal pooling Layer

In previous work Socher et al. [2011] used a dynamic pooling layer to obtain features independent of input size, for 1D signals. Here a variation, which takes into account the 2D nature of images and produces spatially distributed representations at several resolutions following the work of Lazebnik et al. [2006], is presented. It is important to introduce this type of pooling to MPCNN because its output does not depend on the input image size but rather on the number of maps and levels used in the quad-tree. A conventional max-pooling operation would in fact reduce the input image size by a constant factor. While this is fine for convolutional layers, as their operation does not depend on the actual image size, when the classification layer of a MPCNN is reached (e.g. when maps are converted to a single vector) we would have different size vectors which do not fit the pre-defined input dimensions for the MLP.

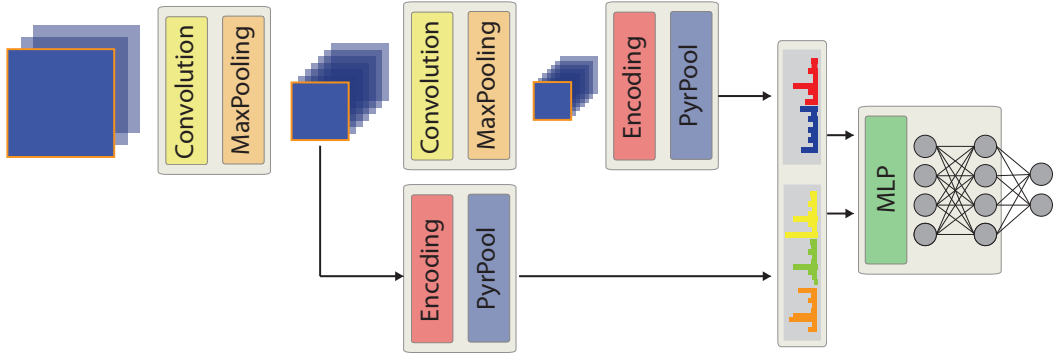


Figure 4.1. Schematic representation of a MSPyrPool Network where histogram-like representations are extracted at two levels and at two scales. The first scale represents the output of the convolutional layer whereas the second scale is given by the output of a pooling (downsampling) layer. The resulting features are concatenated and used as input for the classification layer.

Let us consider a layer with  $k$  maps. If the values within each map are summed, a  $k$ -dimensional feature vector which does not depend on the actual input image size, is obtained. This produces a representation that only depends on the number of maps producing a system which no longer requires fixed size images. Figure 4.2-(a) shows this case which corresponds to the bottom level of the pooling pyramid where a single tile is generated. This already represents a major improvement to obtain a generic steel defects classifier. However, summing all the activations of a map results in higher values for bigger images. To obtain a more stable measure, average pooling is usually preferred. An even more effective way of performing feature pooling uses the max operator instead of the average. This avoids normalization all together and in our experiments always speeded-up learning. Hereafter we consider only pyramidal pooling layers with max-pooling.

To produce a spatially localized feature vector the image is divided into a fixed number of tiles, whose size is dynamically adjusted image by image. Pooling is then performed for each of the produced tiles. The final representation is obtained concatenating all these partial pooling results into a single vector, just as in BoF and is schematized in Figure 4.2-(b) for the case of 4 and -(c) for 9 tiles.

Using a sufficient number of regions Pyramidal Pooling is equivalent to max-pooling and can therefore be considered as its generalization. Let us for example consider the case of  $10 \times 10$  images and max-pooling of  $2 \times 2$ ; this clearly equals

to a Pyramidal Pooling with 5 tiles in each dimension.

The Pyramidal Pooling layer does not have any tunable parameters but, in order to train the feature extraction layers that precede it, the partial derivative of its output w.r.t. its input is required. Let us denote by  $x$  the 3-dimensional input vector of images (e.g.  $[\text{\#rows}, \text{\#cols}, \text{\#maps}]$ ). During the forward pass max-pooling keeps only the maxima values in non overlapping sub-regions of  $x$ , therefore down-sampling the images by a constant factor. The backward pass places the delta values (results of partial differentiation by applying the chain-rule) at the location at which the maxima was found, up-sampling to the original input size.

In the pyramidal pooling layer the forward pass is equivalent to applying a subsampling operation at each level of the pyramid and then concatenating the result into a single output vector as depicted by Figure 4.2. Consequently the backward pass sums over the back propagation of each of the pyramid levels.

Let us denote with

$$\text{forward : } \xi_l(x) \quad \text{backward : } \frac{\partial \xi_l(x)}{\partial x} \quad (4.4)$$

the conventional pooling operations where pooling size is fixed at  $\text{size}(x)/l$ , producing  $l^2$  sub-regions.

The forward pass of a Pyramidal Pooling layer, where  $\text{cat}$  concatenates a set of vectors, can be expressed as

$$y = \text{cat}(\xi_i(x)) \forall i \in \mathcal{L} \quad (4.5)$$

where  $\mathcal{L} = \{l_0, l_1, \dots, l_k\}$  is the set of pooling regions to be used to build the image pyramids. The back-propagation pass can then be expressed by

$$\delta_x = \sum_{i \in \mathcal{L}} \frac{\partial \xi_i(x)}{\partial x}. \quad (4.6)$$

Please note that this framework is however not limited to a max-pooling sub-sampling operation but generalizes to any pooling function.

#### 4.2.2 Multi-scale extraction

A pyramidal feature extraction, is already an improvement over standard MPCNN, as it allows to relax the constraint on a fixed input size. However, it extracts features corresponding only to a single scale (e.g. the nominal size of the “simplified” image where the pooling is performed) and, if applied to the last layer, it will not overcome the multi-scale issue of MPCNN.

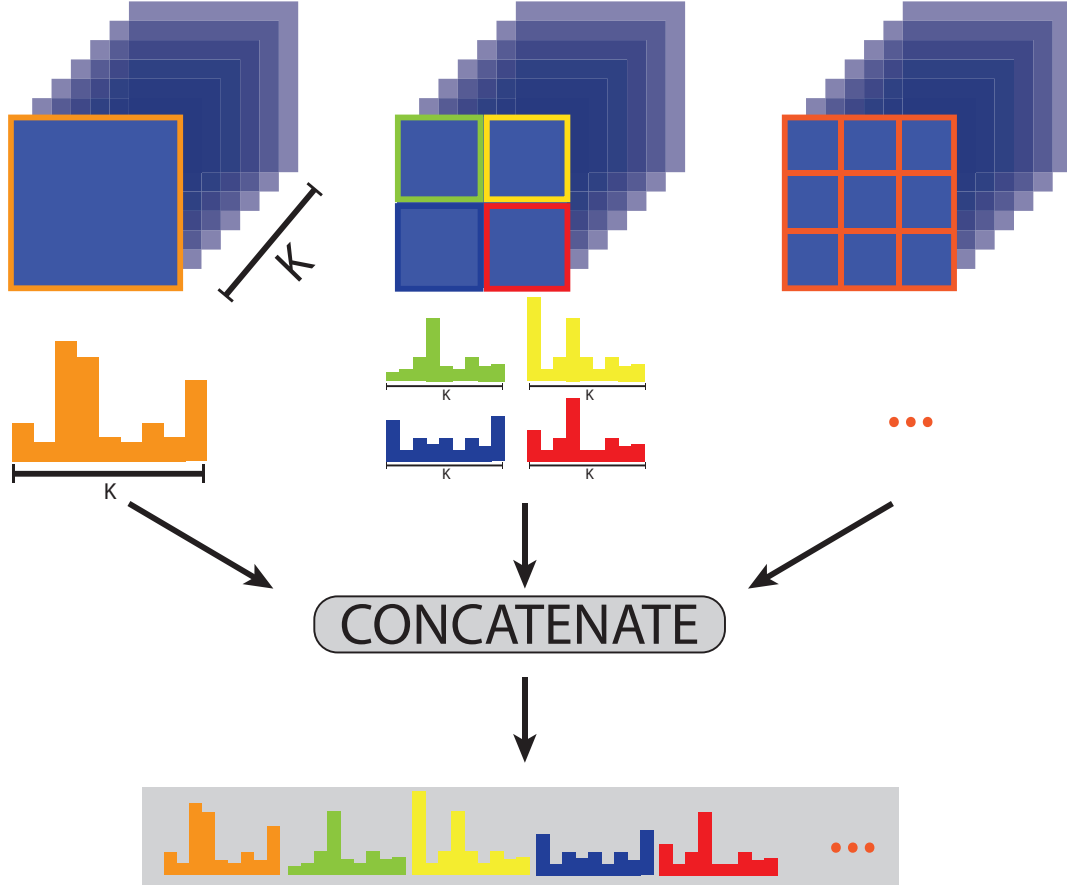


Figure 4.2. Pyramidal Pooling Layer. Features at each level  $l$  of the pyramid are pooled along  $l^2$  equally sized quadrants and the histogram-like representations are concatenated to form a feature vector.

The multi-scale pyramidal feature extraction is implemented by attaching a pyramidal pooling layer for each representation (i.e. layer in the network), and then concatenating the various feature vectors for the classification stage. After a subsampling layer the image gets down-sampled by a constant factor and attaching a pyramidal pooling before and after a max-pooling operation therefore delivers a Multi-Scale feature extraction, as visually illustrated in Figure 4.1 where we get the output of the two convolution and pooling blocks of the MPCNN.

This way if thin features are visible only at the earlier levels of the network they will be short-circuited to the final classification layer and also, broad defects visible only at the top layers will not be affected. We believe it is highly beneficial

to have such rich representation as some defects may exhibit both thin and broad structures and only by combining low and high level features can be successfully recognized.

### 4.2.3 Feature encoding layer

The next step to extend the MPCNN to work well on a limited number of samples as BoF systems is represented by introducing a feature encoding layer. The idea here is to strongly *sparsify* the rich and overly redundant convolutional representation so that it prevents overfitting when scarce number of samples are used.

If we consider the simplest of the feature quantization algorithms,  $k$ -means with hard assignment (VQ), we note that such an algorithm can be approximated by a correlation based measure as the following equality,

$$\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_i (x_i - y_i)^2} = \sqrt{\sum_i x_i^2 + \sum_i y_i^2 - 2 \sum_i x_i y_i}, \quad (4.7)$$

which, in the case of  $\mathbf{x}$  and  $\mathbf{y}$  normalized to have zero mean and unit variance, this reduces to the correlation between  $x$  and  $y$  as their sum will be almost constant. Such an encoding is the *extreme* sparse coding approach as only a single basis is used to fit the input data. This allows us to derive a differentiable approximation of such coding scheme which we name as MLPDict whose details are given below. The reader should note that the approach is general and can be extended with arbitrarily complex functions such as LLC.

**MLPDict Layer.** A fully connected layer with max-pooling winner-take-all units is used to mimic the behavior of the  $k$ -means coding as in VQ of eq. 4.1. The projection of  $x$  with  $\mathbf{W}$ , the weights of the encoding layer, is the correlation between  $x$  and each column of  $\mathbf{W}$ . Taking the maximum approximates the VQ coding scheme. Compared to the BoF approach,  $\mathbf{W}$  serves as the dictionary, however an adaptive one which is tuned sample after sample. Performing a pyramidal pooling operation on the result of such an encoding will produce a histogram-like representation.

Figure 4.3 shows a schematic representation of the encoding layer. The hidden representation of a convnet,  $h$ , is composed of  $K$  images; we consider each pixel as a  $K$  dimensional feature vector (extracted from the densest grid). MLPDict reshapes  $y$  into a matrix with as many rows as pixels ( $\#rows \times \#cols$ ) and as many columns as images ( $\#maps$ ). Applying a fully connected MLP layer



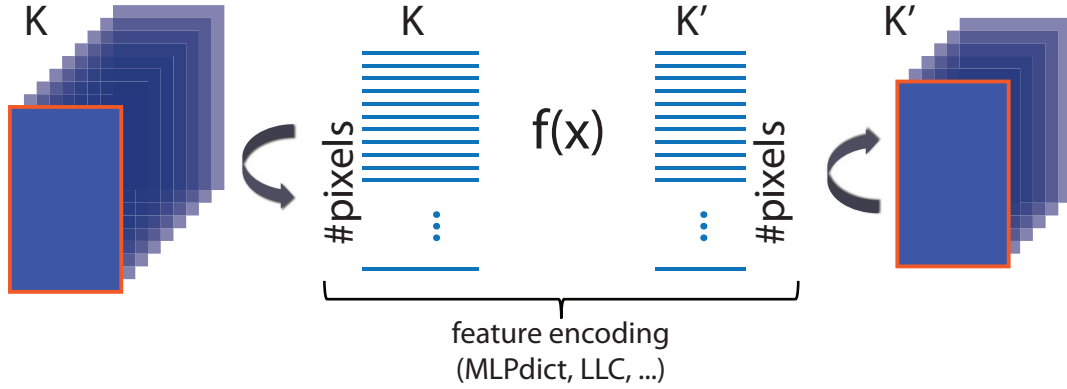


Figure 4.3. The MLPdict layer used for feature encoding. Image responses of a network layer are reshaped to produce  $K$  dimensional feature vectors, where  $K$  represents the number of images in the layer. Each pixel descriptor is mapped into another representation of size  $K'$  for which only the maxima value per row is preserved.

with a weight matrix  $\mathbf{W} \in \mathbb{R}^{K \times K'}$  to the reshaped matrix  $h \in \mathbb{R}^{N \times K}$  will result in  $h' \in \mathbb{R}^{N \times K'}$ . This is reshaped back onto  $K'$  images, where  $N$  corresponds to the number of pixels in each image. When  $K' \ll K$  it acts as a feature selection layer which reduces redundancies; common strategy in image processing, especially in hyper-spectral data processing. When  $K' \gg K$  the layer, thanks to the max-pooling operation, acts as a conventional VQ encoding layer. This is what is used in all the experiments; however the approach is general and extends to any feature encoding algorithm.

As for any layer a nonlinear activation function can be used at the MLPdict output; in such cases we refer to a nonlinear MLPdict layer.

## 4.3 Results

In the experimental evaluation, unless stated otherwise, the average per-class accuracy is utilized to establish the classification performance, a more meaningful measure in case of unevenly distributed datasets. No additional preprocessing, such as translation or deformation is used because of the aforementioned requirements of steel industry. In fact steel defects are not translation, rotation or scale invariant and synthesizing images with conventional data augmentation techniques would introduce such invariances, thus decreasing performance. Carefully designed techniques must be developed to augment or generate plau-

sible defect samples in this domain and ArcelorMittal intends to investigate it in future work.

All nets are trained using stochastic gradient descent (mini-batch of 1) with initial learning rate of 0.001, annealed by a factor of 0.97 at every epoch and momentum term of 0.9. Softmax activation is used at the output layer and the Multi-Class Cross-Entropy (MCCE) loss is minimized

$$\text{MCCE} = - \sum_{i=1}^k (t_i \ln x_i), \quad (4.8)$$

where  $x_i$  is the  $i$ -th output of the network and  $t_i$  is the 1- $k$  coded vector of labels.

We validate the MSPyrPool model first on publicly available benchmarks to compare with other similar published approaches. We then show results on a challenging dataset from the steel industry where our MSPyrPool framework can be applied directly and where convnets fail.

Comparing a MSPyrPool architecture with that of a MPCNN is not an easy task as the two approaches differ considerably. Nevertheless we try to make the comparison as fair as possible by taking equally sized convolutional and sub-sampling layers in both architectures and letting them differ for the choice of the encoding and classification stages. The two systems equals for a particular choice of the MSPyrPool parametrization, namely when the encoding layer is constrained to the identity and the pooling regions are taken at a single resolution (e.g. the one which produces pooling regions of desired size).

**GPU Implementation** The proposed framework requires lot of computational resources to train. MSPyrPool is implemented on GPU using Arrayfire [[AccelerEyes, 2012](#)] for which we experienced speed-ups in the range 15 – 40×, without losing flexibility and ease of coding.

#### 4.3.1 Conventional Benchmarks

Three common evaluation datasets are selected: digit, texture and object recognition. All of them belong to quite orthogonal domains for which ad-hoc techniques are usually applied. In particular the latter one shows clearly the advantage of a MSPyrPool network w.r.t. MPCNN in the context of fully supervised classification. No particular tuning of the architecture has been made as the aim of this section is to show the relative improvement of MSPyrPool nets over MPCNN.

## MNIST

As a reference to compare our approach with a conventional MPCNN we take the well studied MNIST [LeCun et al., 1998] benchmark of handwritten characters. Exemplar images are shown in Figure 4.4. Convnets, regardless on the particular choice of the constituent layers excel on this dataset where all digits are of equal size and centered in the middle of a  $28 \times 28$  grey-scale image. For this experiment the overall classification accuracy is used to easily compare with other published methods.

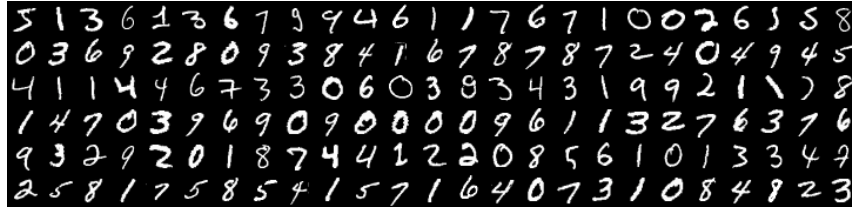


Figure 4.4. Some exemplar images from the MNIST digit recognition dataset.

Table 4.1. Classification results for the MNIST benchmark. The MSPyrPool network is compared with other CNN-based approaches which do not use any input preprocessing.

	Test %
CNN LeNet-5 [LeCun et al., 1998]	99.05
CNN + pre-training [Ranzato et al., 2006b]	99.40
CNN + pre-training [Masci et al., 2011a]	99.29
MSPyrPool	99.13

We use a MSPyrPool net with a convolutional layer with  $5 \times 5$  filters and 100 output maps ( $C \ 5 \times 5 \times 100$ ), a  $2 \times 2$  max-subsampling layer (MP  $2 \times 2$ ), a pyramidal pooling with linear MLPDict at the output of the two subsampling layers with  $l_1 = \{1, 2, 4, 8\}$  and  $l_2 = \{1, 2, 4\}$  quadrants respectively. The resulting feature vector is of size 10600, obtained as the sum of  $l_1^2 * 100$  and  $l_2^2 * 100$ , because we have 100 output maps. Results are shown in Table 4.1. It is interesting to note that the proposed approach is the best among the fully supervised MPCNN approaches and on-par with the ones which use unsupervised pre-training while just using a very small single layer network and 100 filters. We attribute this

to the ability of the winner-take-all strategy used in the encoding layer, which enforces the sparsest of the solutions with one active basis per feature, and to the pyramidal pooling layer which considers the same features at several spatial resolutions.

### CUReT

The Columbia-Utrecht (CUReT gray) database [Dana et al., 1999] contains 61 textures; each with 205 images obtained under different viewing and illumination conditions. Some exemplar images are shown in Figure 4.5. Results are reported for all 61 textures.

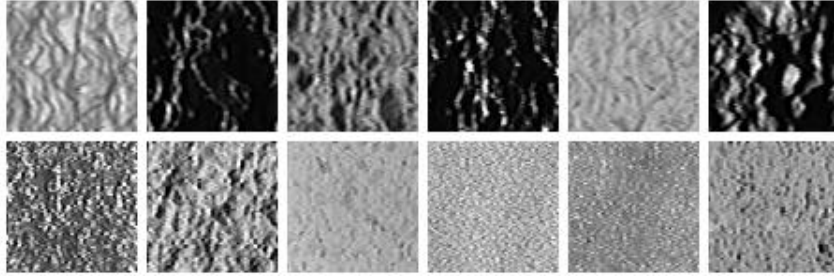


Figure 4.5. Some exemplar images from the CUReT texture recognition dataset.

For training the MSPyrPool network only a single image is required as input, just as in previous work [Varma and Zisserman, 2005], with no information (implicit or explicit) about the illumination and viewing conditions. The conventional evaluation protocol of Varma and Zisserman [2005] was employed, but with a different random split of the data. Every image is normalized to have zero mean and unit variance, a common technique which helps to compensate for very different light conditions.

We train a MPCNN with 5 hidden layers: C  $11 \times 11 \times 20$ , MP  $5 \times 5$ , C  $9 \times 9 \times 20$ , MP  $5 \times 5$ , classification layer.  $\tanh$  is used as activation for every convolutional layer. We compare the result with a MSPyrPool with  $\tanh$  pyramidal pooling MLPDict and codebook size of 100 at the output of the first and second convolutional layers, keeping the rest of the network topology. Features are pooled using  $l_1 = \{1, 2, 4\}$  and  $l_2 = \{1, 2, 3\}$  levels, thus producing a 3500 dimensional vector which is fed to the final softmax classifier.

Table 4.2 compares the MSPyrPool network with the conventional architecture. The MSPyrPool net generalizes much better to the unseen test data and

shows the superiority over conventional MPCNN for the task of texture classification, a domain closely related to steel.

We also train the same model without the MLPDict layer to further show that the encoding stage is important and delivers indeed non-marginal improvements in the final classification performance. The approach reaches 99.0% recognition rate on all 61 classes and greatly outperforms bank-of-filter and Texton (96.4% [Varma and Zisserman, 2005]), whose similar architecture has *pre-wired* feature extractors. This empirically shows that the novel layers, contribution of this work, help to learn better features, and are a valuable addition to the MPCNN framework.

Table 4.2. Classification results for the CURET benchmark. A conventional MPCNN is compared with our MSPyrPool network. We also show the relative improvement of a MLPDict encoding stage.

	Test %
Textons [Varma and Zisserman, 2005]	96.4
MPCNN	96.5
MSPyrPool (no encoding)	93.8
MSPyrPool	99.0

### Caltech101

A further validation of the proposed system is performed on a classical pattern recognition benchmark, Caltech101 [Fei-Fei et al., 2004], where fully supervised MPCNN have seldom successfully been applied [Jarrett et al., 2009]. Some exemplar images of this dataset are shown in Figure 4.6. Usually ad-hoc preprocessing stages and tailored non-linearities are required to obtain a satisfactory performance. MSPyrPool results were compared to those obtained with the similar system which uses conventional BoF with VQ [Lazebnik et al., 2006]; 30 images per class were used for training and testing was performed on at most 50 of the remaining images converted to grey-scale. The feature encoding layer we adopted in this paper is in fact an approximation of the  $k$ -means quantization with hard assignment used in the original work of Lazebnik et al. [2006].

We consider a net with and without an encoding layer. Both nets are composed by: C  $16 \times 16 \times 100$ , MP  $5 \times 5$ . For the net without an encoding layer a pyramidal pooling layer with  $l = \{1, 2, 4\}$  is used to create a 2100-dim feature



Figure 4.6. Some exemplar images from the Caltech101 texture recognition dataset.

vector. For the net with an encoding layer, we used a dictionary size of 1024 (we did not optimize for size and took a fairly large dictionary) just before the pyramidal pooling layer. Using  $l = \{1, 2, 4\}$  results in a 21504-dim feature vector. We train both a net with a linear and non-linear activation function in the encoding layer. For the sake of completeness we also train a MPCNN consisting of: C  $16 \times 16 \times 100$ ; MP  $5 \times 5$ ; C  $13 \times 13 \times 100$ ; MP  $5 \times 5$ ; fully connected classification layer. This MPCNN architecture is by no means the best architecture for this task, and is only listed to quantify the improvement using MSPyrPool nets. Results of all experiments together with results from the literature are listed in Table 4.3.

MSPyrPool nets clearly improve recognition rate compared to a similar sized MPCNN which tend to immediately overfit in just couple epochs.<sup>2</sup> Strong regularizations and normalizations need to be adopted to boost recognition [Jarrett et al., 2009].

Using an encoding layer improves generalization performance even though the resulting nets have many more free parameters. In this experiment using a non-linear activation degrades generalization, probably due to the fact that the dictionary size is too big for the non-linear case, and better results might be obtained using much smaller dictionaries in the encoding layer.

<sup>2</sup>Better results are obtained with deeper and bigger net, we got 40% with a large MPCNN still worse than MSPyrPool.

Table 4.3. Classification results for the Caltech101 benchmark. A MSPyrPool net without an encoding layer (MSPyrPool1), with a linear (MSPyrPool2) and a nonlinear encoding layer (MSPyrPool3) are listed.

	Test %
MPCNN	25.2
MSPyrPool1	52.8
MSPyrPool2	58.0
MSPyrPool3	55.2
Spatial Pyramid [Lazebnik et al., 2006]	64.6
LLC [Wang et al., 2010a]	73.4

Results show that we are able to jointly learn the feature extraction, the quantization and the classification stages fully online in a particularly difficult domain where unsupervised pre-training is required in most ML systems. We also see that the encoding stage does not match the performance of the CV system, perhaps due to the single layer architecture adopted to mimic that of the SIFT descriptors.

#### 4.3.2 Steel-Defects Industrial Benchmark

For this experiment we use a proprietary dataset of ArcelorMittal from a hot-strip mill production line containing 30 different defect classes, the original dataset from which a subset of defect classes has been used in Chapter 3. A region-of-interest (ROI) is provided for each of the instances which vary greatly in size from a minimum edge length of  $\approx 20$  to a maximum of  $\approx 2000$  pixels (please refer to Figure 4.7 for a visual comparison). Furthermore the dataset is unevenly distributed w.r.t. the number of samples per class; a task where the full potential of a MSPyrPool model can be appreciated.

In order to obtain a good support to perform the pyramidal pooling we add background information to get a minimum patch size of 100 pixels along each dimension whenever possible and zero-pad otherwise. We also limit the maximum size per dimension to 500 pixels to accelerate training.

The MSPyrPool was compared to a set of classifiers trained on commonly used features using the same evaluation protocol described in Chapter 3, but with a larger MLP for classification (500 units) as the task is harder and the number of defect classes is much larger. We use a MSPyrPool with: C  $11 \times 11 \times 20$ ,



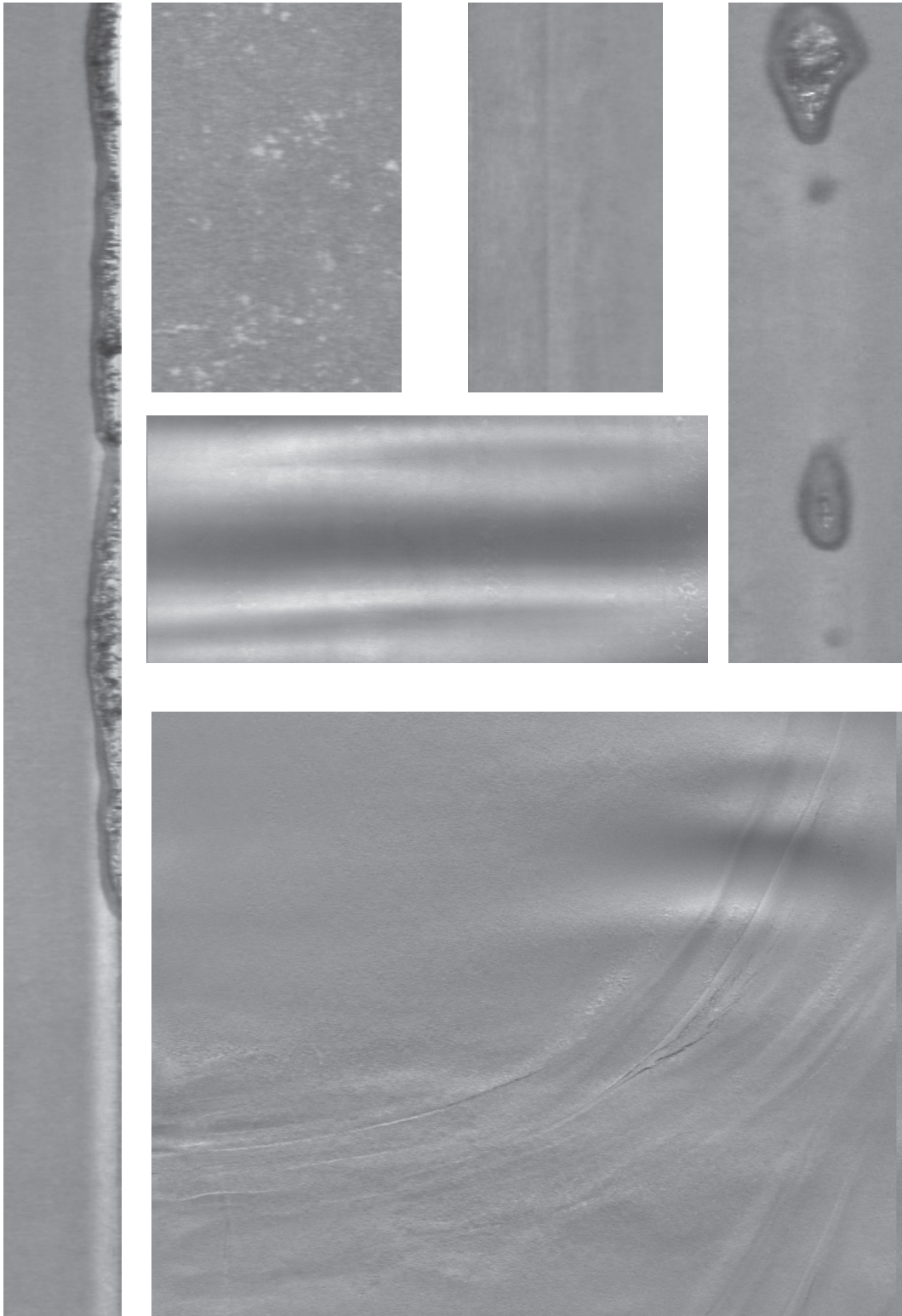


Figure 4.7. Subset of images from the steel-defects benchmark showing the great difference in size among various samples. In this setting is not possible to resize the images to the same size, hence a MPCNN is not applicable to solve this task.



Table 4.4. Classification results for the Steel-defect benchmark where MPCNN fail. Various classifiers, trained on conventional features, are compared with the MSPyrPool network which outperforms, by large margin, any classifier based on engineered features.

	Test %
LBP	29.1
LBP-HF	48.9
MONO-LBP	64.0
VAR	34.3
HOG	60.5
PHOG	58.9
Committee of classifiers	72.1
MPCNN	-
MSPyrPool	75.3

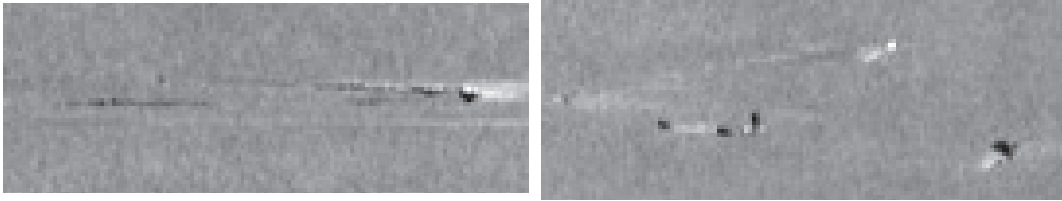


Figure 4.8. Two misclassified images for which the network inverted the corresponding classes (off diagonal elements in the confusion matrix). This example clearly shows the extreme difficulty of this task.

MP  $2 \times 2$ , C  $9 \times 9 \times 20$ , classification layer.  $\tanh$  activation is used for every convolutional layer. Pyramidal pooling layers with  $\tanh$  MLPDict and codebook size of 100 are attached at the output of the first and second convolutional layers. Features are pooled respectively at  $l = \{1, 2, 4\}$  and  $l = \{1, 2, 3\}$  levels producing a vector of 3500 dimensions.

Results are summarized in Table 4.4. We see that a MSPyrPool net can be applied to solve a problem where conventional MPCNN can not. In this setting resizing will destroy the images and remove the actual defect size information. We also see that the performance of the proposed model, which learns everything from pixel representation with no prior knowledge, outperforms any en-

gineered feature classifier by large margin, including PHOG which performs a similar pooling strategy. Even when all features are combined into a very powerful committee of classifiers our single model, with fairly reduced number of parameters and computational complexity, is still achieving the better performance with a relative improvement of 22%.

## Chapter 5

# Fast MPCNN image segmentation and detection

This chapter focuses on MaxPooling Convolutional Neural Networks applied to image segmentation and detection problems. While a lot of work has been devoted to making use of such models for these tasks, performance improvements have been achieved mainly thanks to recent advances in hardware; rather than from fundamental modifications or extensions to the basic architecture.

MPCNN, in this thesis have been so far used only to predict a class label for a given input image. For image *classification*, in fact, MPCNN return a vector of class posterior probabilities when provided with an input image of fixed width and height.

For *segmentation*, however, one of such posterior probability vectors must be assigned to each of the input image pixels. To achieve this goal the best solution is to take pixels within a square patch of odd size and classify them according to the class of its central pixel. Please note that patches are odd in size so that there is no ambiguity in determining the center as it corresponds to the median; any other convention could be used though as far as it is consistently adopted. The network is trained as usual, but on patches extracted from a set of images with ground-truth segmentations (i.e. the class of each pixel is known). Once trained, the MPCNN segments an unseen image by simply classifying all of its pixels. This produces an output image where pixels are replaced either by class labels or posterior probabilities of each pixel belonging to a particular class.

Finally, *object detection* within an image is trivially cast as a segmentation problem: pixels close to the centroid of each object are classified differently from background pixels and the posterior probabilities are transformed into disjoint object regions. This approach is the gold standard for person detection [[Dalal](#)

and Triggs, 2005] and forms the basis of the winning strategy of recent record-breaking results [Ciresan et al., 2013b; Sermanet et al., 2013a]. Main difference the learned feature extraction with a MPCNN. Even the computer vision community, usually resilient to such architectures, has now started to use them to improve their detection systems [Girshick et al., 2013].

Solving segmentation and detection tasks requires to apply the network to every patch contained in the image, which is prohibitively expensive when implemented in the naïve, straightforward way. Consider, for example, a net with a convolutional layer immediately after the input layer: when evaluating the first patch contained in the input image, the patch is convolved with a large number of kernels to compute the output maps; when evaluating the next (typically overlapping) patch, such convolutions are re-evaluated resulting in a huge amount of redundant computation. A better approach is to compute each convolution only once for the whole input image. In Figure 5.1 such problem is visually exemplified; the yellow and blue patches are input to the same MPCNN to predict their respective posterior probabilities. However, after the first convolutional layer the internal representation of the network for the two inputs is highly redundant as only a single column has changed. Similar behavior can be found in deeper layers. Even resorting to efficient GPU implementations [Ciresan et al., 2011b; Bergstra et al., 2010; Collobert et al., 2011; Krizhevsky et al., 2012] does not solve this scalability issue because it does not affect the amount of computations to be performed but simply how fast they can be done.

In the particular case of a CNN without max-pooling layers, this optimization is trivially implemented by computing all convolutions in the first layer on the entire input image, then computing all convolutions in subsequent layers on the resulting maps. This approach of Farabet et al. [2011, 2009] yielded real-time detection performance when combined with dedicated FPGA or even ASIC integrated circuits. Similar approach has been followed in the work of Turaga et al. [2009, 2010]; Ning et al. [2005]; Jain et al. [2007] to successfully segment images.

However, present convnets owe much of their power to max-pooling layers interleaved with convolutional layers. Max-pooling cannot be handled using the straightforward approach outlined above. For example, when we perform a  $2 \times 2$  max-pooling operation on a set of maps, we obtain smaller maps which do not contain information from all the input pixel; instead, only patches whose upper left corner lies at odd coordinates of the original image are represented. Any subsequent max-pooling layer would further worsen the problem. This issue is shown in Figure 5.1 where after the max-pooling layer the information contained in the output maps is completely disjoint.

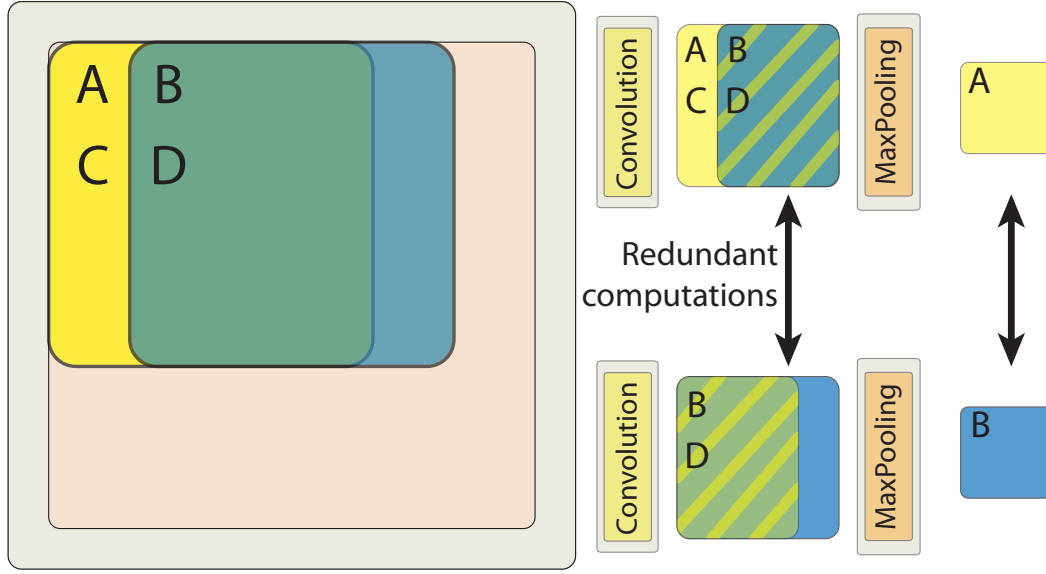


Figure 5.1. Illustrative example of how much redundant computation there is when testing a MPCNN on a patch-by-patch basis. Similarly such situation is encountered also during training. Yellow and blue patches are input to the same MPCNN which computes twice the same intermediate results indicated by the diagonal stripes. When a max-pooling operation is applied, given its partial result it is possible to classify only the pixel indicated in the corresponding output; e.g. A for yellow and B for blue.

This chapter<sup>1</sup> presents two state-of-the-art variants of MPCNN developed to approach image segmentation and detection which approach near-optimal resource usage.

The first contribution, in Section 5.1, shows how to efficiently apply a trained MPCNN on all patches of an image. The second contribution, in Section 5.2, shows how to perform training on the entire image without resorting to the patch tiling approach via a novel MaxPooling Fragment (MPF) layer. It can process images order of magnitudes faster than previous approaches, even when comparing CPU and GPU implementations. We refer to a MPCNN with MPF layers as SEGNN.

The advances presented here allow MPCNN to run at speeds that are much closer to those of real time inspection systems. Additionally, our contribution made possible to efficiently use convnets for the task of object detection, as in

<sup>1</sup>This chapter is based on [Giusti et al. \[2013\]](#); [Masci et al. \[2013b\]](#).

OverFeat of [Sermanet et al. \[2013a\]](#), a brilliant and freely available tool which provides fast scanning of a state-of-the-art convnet trained on ImageNet [[Deng et al., 2009](#)]. Similarly to what happens with steel, a detector trained on large patches needs to be raster scanned on all positions in the input image to produce output probabilities of a given target object to be present at a given location in an image.<sup>2</sup>

## Notation

The following notation is adopted throughout the rest of this chapter. The set of training images is indicated by  $\mathbf{X}$ ; the corresponding ground-truth annotations by  $\mathbf{T}$  (thus mapping a class to each pixel);  $x_i$  and  $t_i$  refer to a particular training and target image, respectively. This means that each  $x_i$  has a corresponding  $y_i$  of same size where each pixel in  $y_i$  indicates the class label of the corresponding pixel in  $x_i$ . The net is parametrized by  $\theta$ , the union of all parameters of all layers. The objective function of the minimization problem is denoted  $L(\xi(\mathbf{X}), \mathbf{T}; \theta)$  where, as in eq. 2.1,  $\xi(\mathbf{X})$  is the network mapping applied to the set of input images  $\mathbf{X}$ . A layer  $j$ , indicated by  $\xi^j(x_i^j)$ , is a function mapping input storage to output storage.

Such a storage is defined as the set  $\mathbf{F} = \{\cup_{i=1}^N f_i\}$ , where each  $f_i$  represents a stack of maps, here denoted as *Fragments*. For example, the input layer will be a storage  $\mathbf{F}^{\text{input}}$ , with cardinality 1.  $\mathbf{F}^{\text{input}}$  contains a single fragment  $f_0^{\text{input}}$ , corresponding to the input image. More generally we indicate by  $\mathbf{F}^l$  the storage at level  $l$  in the hierarchy and by  $|\mathbf{F}^l|$  the number of fragments it contains. Each fragment  $f_i \in \mathbf{F}^l$  has size  $s_x^{l,i}$  rows,  $s_y^{l,i}$  columns and  $s_z^{l,i}$  number of maps. As the number of maps in each fragment is constant we will use the notation  $s_z^l$  to indicate the  $z$ -dimension of a generic fragment in layer  $l$ . When images are square, to reduce clutter, we denote by  $s^l$  the size of either the height or the width.

This architecture strongly differs from conventional MPCNN by the choice of storage structure. In standard models every storage is a container of a stack of maps. Instead, in our case, the same data is necessarily split into a number of fragments, each of which is a stack of maps. Hence, our architecture equals a conventional MPCNN when every fragment has cardinality equal to one.

---

<sup>2</sup>A free implementation, different from the one used in the experiments, in Theano [[Bergstra et al., 2010](#)], has been released so that users can benefit from these innovations more easily.

## 5.1 Fast Image Scanning

In what follows we present the first contribution, an optimized forward-propagation approach which avoids the aforementioned problems by fragmenting the output of each max-pooling layer, such that each one contains information independent of the others, and the union of all fragments contains information about all the patches in the input image. A similar approach was previously used by [Nasse et al. \[2009\]](#) for handling a single subsampling layer in a simple convnet for face detection. Our mechanism, however, is completely general. It handles arbitrary architectures mixing convolutional and max-pooling layers in any order, and ensures that no redundant computation is performed at any stage.

Let us now consider a square gray-level input image of size  $s \geq w_0$ . We want to compute the network outputs on all patches completely contained within it – i.e.  $(s - w_0 + 1)^2$  patches.

Maps in the same fragment have the same size; maps in different fragments may have different sizes though, and not all such sizes may be square even though the input image is. Here we consider, for simplicity, square input images. Our method, however, is not limited to this case but works with any input image dimensions.

The input image is provided as a single fragment with index 0, therefore  $|F^0| = 1$ . Such fragment contains the set  $\{f_0^0\}$  of square maps with sizes

$$s_x^{0,0} = s \quad (5.1)$$

$$s_y^{0,0} = s \quad (5.2)$$

$$s_z^{0,0} = 1. \quad (5.3)$$

Let  $l$  denote the index of a *convolutional layer*. Its output consists of a set of fragments, such that  $|\mathbf{F}^l| = |\mathbf{F}^{l-1}|$ .

A given fragment  $f \in \mathbf{F}^l$  has size

$$s_x^{l,f} = s_x^{l-1,f} - (k - 1) \quad (5.4)$$

$$s_y^{l,f} = s_y^{l-1,f} - (k - 1), \quad (5.5)$$

where  $k$  is the size of the (square) kernels of layer  $l$  and  $s_z^{l,f}$  is determined at network definition. Again, note that maps in different fragments may have different sizes, but all maps in the same fragment have the same size.

Each  $f_i^l$  is obtained as a function of  $f_i^{l-1}$ , using the same operations as with patch-level forward propagation. However, convolutions are performed on the

(large) extended maps rather than on small maps, like in the patch-level approach.

Let  $l$  denote the index of a *max-pooling* layer. Its output consists of a set of fragments, such that  $|\mathbf{F}^l| = k^2 |\mathbf{F}^{l-1}|$ , where  $k$  is the size of the square max-pooling kernel. In particular, each input fragment  $f^{\text{in}} \in \mathbf{F}^{l-1}$  generates  $k^2$  output fragments.

Consider a given input fragment  $f_i$ , associated with the set  $\mathbf{F}^{l-1}$ . Let  $\mathbf{o}$  be a set of  $k^2$  2D offsets defined as the Cartesian product  $\{0, 1, \dots, k-1\} \times \{0, 1, \dots, k-1\}$ . E.g., for  $k = 2$ :

$$\mathbf{o} = \{(0, 0), (1, 0), (0, 1), (1, 1)\}.$$

For a given input fragment  $f_i^{l-1}$  and for each offset  $o \in \mathbf{o}$ ,  $o = (o_x, o_y)$ , one output fragment  $f_j^l$  is produced. Each of the maps in  $\mathbf{F}^l$  is generated by applying the max-pooling operation to the corresponding extended map in  $f_i^{l-1}$ , by starting at the top-left offset  $(x, y) = o$ . Specifically, the pixel at coordinates  $(\bar{x}, \bar{y})$  in the output map is computed as the maximum of all pixels in the corresponding input map at coordinates  $(x, y)$  such that:

$$\begin{aligned} o_x + k\bar{x} &\leq x \leq o_x + k\bar{x} + k - 1 \\ o_y + k\bar{y} &\leq y \leq o_y + k\bar{y} + k - 1. \end{aligned} \quad (5.6)$$

Then the size of the output maps in  $\mathbf{F}^l$  is given by:

$$s_x^{f,j} = \text{div} \left( (s_x^{l-1,i} - o_x), k \right) \quad (5.7)$$

$$s_y^{f,j} = \text{div} \left( (s_y^{l-1,i} - o_y), k \right), \quad (5.8)$$

where  $\text{div}$  denotes the integer division operation. The max-pooling operation thus ignores the following parts of the input extended maps:

- $o_y$  leftmost columns;
- $o_x$  top rows;
- $\text{mod} \left( (s_x^{l-1,i} - o_x), k \right)$  rightmost columns;
- $\text{mod} \left( (s_y^{l-1,i} - o_y), k \right)$  bottom rows.

**Theoretical Speed-Up** We now discuss the speed-up of our image-based approach in comparison to separate evaluation of all patches contained in the input image. We consider as an example the largest network used in [Ciresan et al.](#)



Table 5.1. Theoretically required FLOPS for convolutional layers when segmenting a  $512 \times 512$  image using patch-based ( $\text{FLOPS}_{\text{patch}}^l$ ) and image-based ( $\text{FLOPS}_{\text{image}}^l$ ) approaches.

Layer ( $l$ )	$s$	$s^{l-1}$	$s_z^{l-1}$	$s_z^l$	$w_l$	$k_l$	$ \mathbf{F}^l $	$\text{FLOPS}_{\text{patch}}^l [\cdot 10^9]$	$\text{FLOPS}_{\text{image}}^l [\cdot 10^9]$	speed-up
1	512	559	1	48	92	4	1	3408	0.5	7114.8
3	512	279	48	48	42	5	4	53271	35.9	1485.1
5	512	139	48	48	18	4	16	6262	22.8	274.7
7	512	69	48	48	6	4	64	695	22.5	30.9
Total								63636	81.6	779.8

[2012a] for neuronal membrane segmentation [Cardona et al., 2010]. The image size (one slice with neuronal tissue data) is  $512 \times 512$  pixels. Its edges are mirrored, to get enough pixels for applying the network to all positions. Mirroring pads the image by adding pixels taken going from the edge towards the inside. Adding zeros would in this case virtually add strong corners which would bias the result. We limit our analysis to convolutional layers, which are by far the most computationally intensive part of a MPCNN. Conversely, max-pooling layers are simple and fast, requiring less than 1% of the computing time in most practical nets.

For the patch-based approach, the required amount of floating-point operations (FLOPS) for computing the convolutions in layer  $l$  when scanning an image by a MPCNN obeys the following formula:

$$\text{FLOPS}_{\text{patch}}^l = s^2 \cdot s_z^{l-1} \cdot s_z^l \cdot w_l^2 \cdot k_l^2 \cdot 2,$$

where  $s^2$  is the number of pixels in the input image, and, for each convolutional layer  $l$ ,  $w_l^2$  the number of pixels of the map, and  $k_l^2$  the number of kernel pixels. The factor “2” reflects that we have one addition and one multiplication for each component of the dot product.

For the image-based approach, the FLOPS can be computed using the following formula:

$$\text{FLOPS}_{\text{image}}^l = s_x^l \cdot s_y^l \cdot s_z^{l-1} \cdot s_z^l \cdot |\mathbf{F}^l| \cdot k_l^2 \cdot 2,$$

where  $s_x^l \cdot s_y^l$  represents the size of a fragment in layer  $l$  (to simplify the formula, we assume all fragments have the same size, although they may differ in size by at most one pixel). For the input layer ( $l = 0$ ), mirroring the borders implies that  $s_x^0 = s_y^0 = s + (w_0 - 1)/2$ .

Table 5.1 reports such computations for all convolutional layers in the large network of Ciresan et al. [2012a]. The patch-based approach requires 779.8

Table 5.2. Speed for segmenting a  $512 \times 512$  image using the large net described in [Ciresan et al. \[2012a\]](#).

Method	Time per image [s]	Speed-up relative to GPU-patch
matlab-patch	24641.54	-
GPU-patch	492.83	1
matlab-image	15.05	32.8

times more FLOPS than the image-based approach.

**Experimental speed-up** This theoretical speed-up was verified experimentally by three different implementations:

- matlab-patch: a plain Matlab implementation of the patch-based approach;
- GPU-patch: a heavily optimized implementation of the patch-based approach running on a GTX-580 graphics card using CUDA;
- matlab-image: a plain Matlab implementation of the image-based approach.

In Table 5.2 we report computation times of the same MPCNN when used to segment a  $512 \times 512$  image for each of the implementations.

Results clearly show that the image-based implementation yields a dramatic speed-up over patch-based approaches. In particular, *matlab-image* yields a 32-fold speed-up when compared to the highly-optimized *GPU-patch* implementation, despite the former being implemented in a slower environment and without attention to low-level optimizations. The impact of GPU and low-level optimizations is obvious as the *GPU-patch* approach is 50 times faster than *matlab-patch*.

## 5.2 Beyond patches: learning on full images

Processing images instead of patches is a great advantage for the testing phase of a MPCNN. However, training still remains the bottleneck when large datasets need to be processed, a requirement of the modern big data-driven economy.

Sampling patches becomes not-trivial and use of all patches prohibitive even with GPUs, especially with large models. One of our contributions addresses such issue and shows how to do efficient training of MPCNN using images as input, therefore removing all redundant calculations. We refer to a MPCNN with MPF layers as SEGNN.

### 5.2.1 The MaxPoolingFragment (MPF) layer

In this section the second contribution of this chapter is introduced; the **Max-PoolingFragment** (MPF). Given an input image  $x_i$ , a conventional  $k \times k$  MP layer produces a smaller image, for which only a single value in each non-overlapping  $k \times k$  neighborhood is kept.

The forward pass of our MPF layer closely follows the detailed description reported in Section 5.1. With a MPF layer there will be  $k^2$  different offsets in the input map, each one producing an output fragment. Thus, if the number of fragments in input is  $|\mathbf{F}^{\text{in}}|$ , we will have  $|\mathbf{F}^{\text{in}}|k^2$  fragments in total. All redundant computations are removed. This equals to a morphological dilate, with a flat structuring element of size  $k \times k$  anchored to the top-left pixel, followed by a downsampling scheme which produces fragments of the image according to their offset within the kernel window.

From a software engineering perspective, an MPF layer can also be seen as a collection of max-pooling layers, one for each generated fragment.

Consider a training image  $x_i$  and its ground truth  $t_i$ . By means of the newly-defined MPF layer, we can quickly forward-propagate the network on the whole image to generate an output image  $\hat{x}_i$ , in which every pixel is replaced by the output of the corresponding MPCNN applied to every patch in  $x_i$ . We can now compute the partial derivative of the loss function  $L$  w.r.t.  $\hat{x}_i$ ; e.g.  $\hat{x}_i - t_i$  when minimizing the mean squared error loss with linear outputs. This is the first step of the back-propagation algorithm.

In the next section we show how to process all these errors without considering each output pixel separately.

### 5.2.2 Back-propagation through an MPF layer

As SEGNN uses fragments, we first have to redefine how a layer processes its input and computes its partial result of back-propagation. This generalizes MPCNN operations as explained in object-oriented Matlab pseudocode by Algorithms 3 and 4.

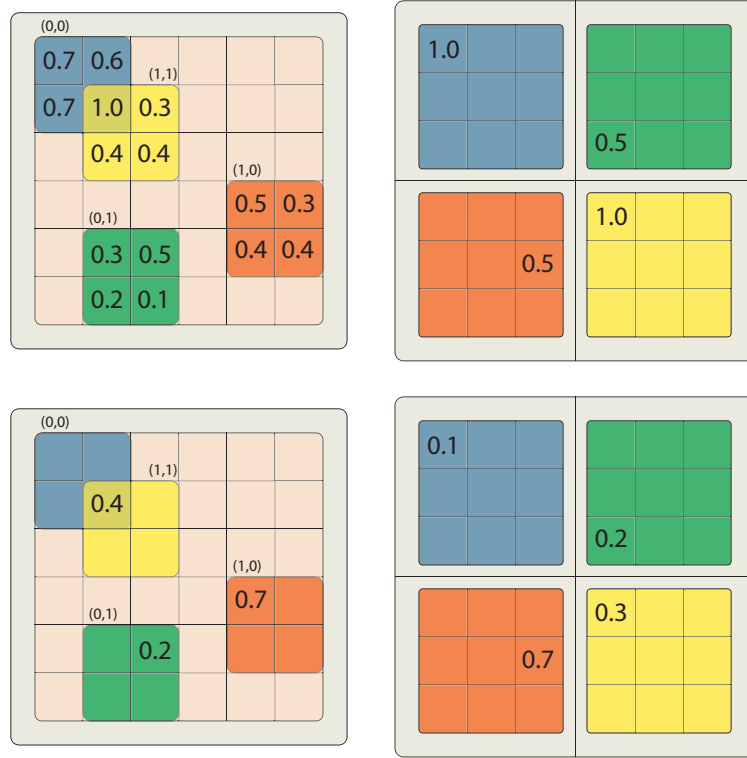


Figure 5.2. MPF layer for the  $2 \times 2$  pooling case. Top: Forward pass, Fragments (0,0) and (1,1) share the same maximal element; Bottom Back-propagation pass where partial derivatives are pushed back to the previous layer in the hierarchy; the partial results of each Fragment are summed together.

Algorithm 3 shows the generic forward pass of a layer in our fragment-based MPCNN framework. Similarly the backward pass is derived. Please note that a layer now produces a set of fragments, collection of maps, instead of a single output map.

Algorithm 4, instead, illustrates the interface for the gradient computation. Because the same input pixel can be used several times to generate more than one output fragment (e.g. the same function is applied to all input fragments during the forward pass) gradients need to be accumulated.

This new neural network interface makes it much easier to derive and implement the backward pass for a MPF layer. Figure 5.2 gives an illustrative example of how a MPF layer works in the  $2 \times 2$  pooling case. The output consists of 4 fragments, each containing as many maps as the input layer, indexed accordingly. We also exemplify the case where the same element (pixel with value 1.0

---

**Algorithm 3:** Pseudocode for the forward pass of a layer the SEGNN framework operating on fragments;  $\cup$  indicates the concatenation of two sets.

---

**Data:** Layer  $l$ , Input storage  $\mathbf{F}^{\text{in}}$   
**Result:** Output storage  $\mathbf{F}^{\text{out}}$   
**for**  $i=1$  **to**  $\mathbf{F}^{\text{in}} \rightarrow n\text{Fragments}()$  **do**  
   $\mathbf{F}^{\text{out}} = \mathbf{F}^{\text{out}} \cup l \rightarrow \text{fwd}(f_i^{\text{in}});$   
**end**

---



---

**Algorithm 4:** Gradient computation pseudocode of a generalized MPCNN layer.

---

**Data:** Layer  $l$ , Input storage  $\mathbf{F}^{\text{in}}$ , Partial result of back-propagation  $\mathbf{F}^{\delta}$   
**Result:**  $g: \frac{\partial L(\Theta)}{\partial l \rightarrow \text{params}}$   
**for**  $i=1$  **to**  $\mathbf{F}^{\delta} \rightarrow n\text{Fragments}()$  **do**  
   $g = g + l \rightarrow \text{grad}(f_i^{\delta});$   
**end**

---

in Figure 5.2–top) is the maximum for two different offsets of the pooling kernel (respectively (0,0) and (1,1)), generating the corresponding output fragments (respectively blue and yellow). During back-propagation the partial results of differentiation, shown in Figure 5.2–bottom–right, are processed for every fragment and then summed together. The subsequent convolutional layer processes the 4 fragments through the interface of Algorithm 3, as shown in Figure 5.3, and computes the gradient by summing gradients of 4 fragments. Pseudocode for both operations is shown in Algorithms 5 and 6.

In Algorithm 5 we show the forward pass of a MPF layer. For every input fragment a set of fragments is produced, one for each offset in the pooling kernel. The output is composed by their union as already illustrated in Figure 5.2. With MP we indicate the usual MPCNN downsampling operation applied to the fragments  $\mathbf{F}^{\text{in}}$ . The layer produces an additional output as well, **mpIDXs**, where indices of maxima values in the input maps are stored. This makes easier and faster the backpropagation phase.

After all fragments are computed the classification stage is attached so that a prediction can be assigned to each of the pixels. Similarly to a MLPDict layer (Section 4.2.3) each map is reshaped to have as many rows as pixels and as many columns as channels. Then, the same classifier is used for each of the fragments as for the convolutional layer, following the interface of Algorithm 3.

Algorithm 6 shows instead the pseudocode for the backpropagation step

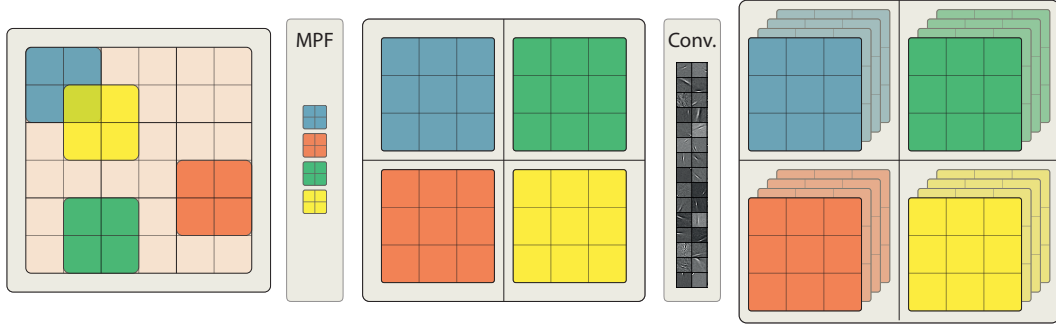


Figure 5.3. Illustrative example of how subsequent convolutional layers of a MPF layer work. The same operation is applied to each output fragment; the gradient is the sum of the gradients of the fragments.

where partial derivatives of the layer's output w.r.t. its input are computed. It produces a set of fragments equal to the one of the input layer  $\mathbf{F}^{\text{in}}$  which was used in Algorithm 5. The function `MP_bkp` places the partial result at the correct position indicated by `mpIDXs`.

---

**Algorithm 5:** Forward pass of an MPF layer. See text for details.

---

**Data:** Input storage  $\mathbf{F}^{\text{in}}$ , Pooling kernel  $P$

**Result:** Output storage  $\mathbf{F}^{\text{out}}$ , MP indices `mpIDXs`

$\mathbf{F}^{\text{out}} = \{\}$  ;

**for**  $i=1$  **to**  $\mathbf{F}^{\text{in}} \rightarrow n\text{Fragments}()$  **do**

**for**  $j=1$  **to**  $\text{size}(P, 1) * \text{size}(P, 2)$  **do**

$[r, c] = \text{ind2sub}(j, \text{size}(P))$  ;

$[a, b] = \text{MP\_fwd}(f_i^{\text{in}}(r:\text{end}, c:\text{end}), P)$  ;

$\mathbf{F}^{\text{out}} = \mathbf{F}^{\text{out}} \cup a$  ;

`mpIDXs` = `mpIDXs`  $\cup b$  ;

**end**

**end**

---

---

**Algorithm 6:** Back-propagation pass of a MPF layer. Refer to the text for details.

---

**Data:** Input storage  $\mathbf{F}^{\text{in}}$ , Result of fwd  $\mathbf{F}^{\text{out}}$ ,  $P$ , **mpIDXs**

**Result:** Output storage  $\mathbf{F}^{\delta}$

```

for  $i=1$  to  $\mathbf{F}^{\text{out}} \rightarrow n\text{Fragments}()$  do
  for  $j=1$  to  $\text{size}(P, 1) * \text{size}(P, 2)$  do
     $s = \text{findSourceFragment}(i, j)$  ;
     $f_s^{\delta} = f_s^{\delta} + \text{MP\_bkp}(f_s^{\delta}, \mathbf{mpIDXs}_s)$  ;
  end
end

```

---





## Chapter 6

# Application of MPCNN to steel segmentation and detection

This chapter reports the results of applying the methods proposed in Chapter 5 to image segmentation and steel defect detection. The method is first validated on the membrane segmentation benchmark [Cardona et al., 2010] where it retains almost the same accuracy as the original model that won the ISBI 2012 Electron Microscopy competition, while being trained at a much higher speed. Then we investigate several steel industry applications ranging from gray level (Section 6.1) to multivariate images (Section 6.2) for segmentation and detection. Thanks to the fast image scanning for MPCNN we are the first, to the best of our knowledge, to be able to approach industrial real-time feasibility for large models which need to be tested on every region of the input image.

The approach seamlessly scales to large dimensional inputs and does not suffer from the aforementioned problems that afflict other methods, while still requiring almost no prior domain knowledge.

### 6.1 Results on single-channel images

SEGNN was validated on several applications on single-channel images (i.e. gray-level images). First, the segmentation performance and the speed-up are measured w.r.t. a patch-based approach on a membrane segmentation task (Section 6.1.1). Then we evaluate it on two challenging industrial datasets for steel defect detection (Section 6.1.2).

In both applications, networks were trained to minimize the multi-class cross-entropy loss (MCCE) – a commonly-used error function for classification tasks. MCCE is computed by considering each pixel independently as if the network

were to be trained patch-by-patch. In fact, SEGNN, considering all output errors could easily be extended with loss functions which ensure, for example, smoothing of nearby predictions. When using the patch-by-patch training procedure adding such output priors is hard, or even impossible.

The framework is implemented in Matlab, on CPU, and uses Intel Performance Primitives to perform convolutions.<sup>1</sup>

### 6.1.1 Membrane Segmentation

We use the public dataset of the ISBI 2012 Electron Microscopy Segmentation challenge [Cardona et al., 2010]. It consists of a volume of 30 gray level images of size  $512 \times 512$  pixels. Given an image as the one of Figure 6.1–(left) we are asked to provide a segmentation into a binary image so that each pixel is assigned either to the background or to the membrane.

As in previous work [Ciresan et al., 2012a] the rotational invariance of the problem was exploited by synthesizing additional training images that were randomly rotated and flipped along both dimensions. Also, pixels outside of the boundary of testing images are mirrored – which allows to preserve the size of the output. We consider the network architectures N3 and N4 of Ciresan et al. [2012a], which contributed to the top-scoring entry in the challenge.

N3 operates on a  $65 \times 65$  window and has the following structure: C  $3 \times 3 \times 48$ , MPF  $3 \times 3$ , C  $5 \times 5 \times 48$ , MPF  $2 \times 2$ , C  $3 \times 3 \times 48$ , MPF  $2 \times 2$ , FC 100, FC 2.

N4 operates on a  $95 \times 95$  window and has the following structure: C  $5 \times 5 \times 48$ , MPF  $2 \times 2$ , C  $5 \times 5 \times 48$ , MPF  $2 \times 2$ , C  $3 \times 3 \times 48$ , MPF  $2 \times 2$ , C  $3 \times 3 \times 48$ , MPF  $2 \times 2$ , FC 200, FC 2. Please note that in SEGNN the filter size of each convolutional layer, due to implementation choices, must be odd. Therefore, the model here reported may vary slightly from the original up to a row or column in the convolutional kernel.

The networks were trained using stochastic gradient descent safe-guarded by the Armijo rule, updating the weights after every image has been presented to the net. Convergence is reached generally after 100 epochs, when the network has seen 3000 different images, the equivalent of roughly 390 million patches.

Figure 6.1 shows a segmentation example for a test slice. Table 6.1 compares SEGNN to the highly-optimized GPU patch-based approach of Ciresan et al. [2012a] in terms of training times. Although SEGNN runs on CPU in the Matlab environment, it still yields a huge speed-up. At the same time, the

<sup>1</sup>A free implementation in Theano [Bergstra et al., 2010], a very powerful tool for deep learning research, is available at <http://www.idsia.ch/~masci/segnn>

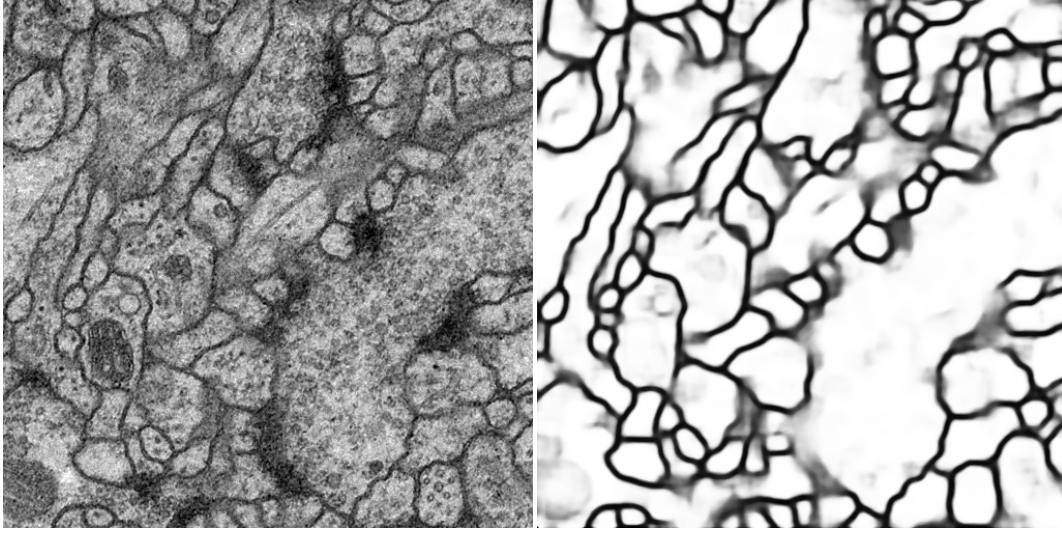


Figure 6.1. A slice of the test set segmented using SEGNN trained on full images. The image on the right shows the probability of each pixel to be assigned to the background (white) or to the membrane (black). Please refer to text for the network architecture details.

difference in segmentation performance is negligible: 6.8% vs 6.6% pixel error rates, for N3, for SEGNN and the one of [Ciresan et al. \[2012a\]](#), respectively. Errors are evaluated directly on the competition server.

Table 6.1. Comparison of training times for the Membrane dataset. The overhead for generating the transformed samples is also included in the overall computation. The relative speed-up of SEGNN is shown in parenthesis.

	Patch (GPU) [Ciresan et al., 2012] patches/s	Image (CPU, Matlab), SEGNN patches/s
N3	260	4500 (17× speed-up)
N4	130	3000 (23× speed-up)

### 6.1.2 Single Defect Detection

We use a proprietary dataset from ArcelorMittal, consisting of 534 images, each with resolution of  $550 \times 240$  pixels. Images are acquired with a matrix camera di-

rectly from a production plant. Illumination is highly variable, and many images are severely under- or over-exposed, which hinders naïve processing techniques. 70 of such samples contain a defect which covers part of each image. This type of defect is very difficult to detect due to its variable and subtle appearance (see Figure 6.2). A ground-truth segmentation of the area (if any) containing the defect of each image is given. We use 50% randomly-sampled images for training, 25% for validation, and the rest for testing.

For this task we are only interested in detecting whether a given image contains a defect of interest. A common application is detection of coil markers. This type of defect is generated when an object gets stuck in the rolling section of the hot-strip mill. Therefore at periodic times, after every revolution of the coil, its shape is impressed on the flowing steel.

The network operates on a  $31 \times 31$  window and has the following structure: C  $7 \times 7 \times 8$ , MPF  $2 \times 2$ , C  $5 \times 5 \times 8$ , MPF  $2 \times 2$ , C  $5 \times 5 \times 8$ , FC 100, FC 2. LBFG-S is used as optimization algorithm because it delivered the best performance. Images are also down-sampled by a factor of 4 to further speed-up learning. The per-class weighted MCCE loss function is employed on this unbalanced dataset,

$$\text{w-MCCE} = - \left( \frac{\sum_{i \in C_0} (t_i \ln x_i)}{|C_0|} + \frac{\sum_{i \in C_1} (t_i \ln x_i)}{|C_1|} \right), \quad (6.1)$$

where  $C_0$  and  $C_1$  indicate the set of input patches belonging to, class 0 and class 1, respectively.

There are, in fact, only very few pixels which correspond to the defect, therefore learning is prone to naïve convergence to solutions which always favor predicting the background. This would produce very poor performance as most of the markers would be considered as background.

Each training epoch takes on average 44s (also accounting for the overhead due to LBFG-S optimization). This amounts to 0.16s per image on a i7-2600 quad-core machine, where the whole system is trained in two hours. Because an image contains roughly 8.2K patches, our system is effectively processing 50K patches per second. Training patch-by-patch is significantly slower even when using highly optimized GPU implementations.

Segmenting a new image requires 0.07s. We believe that the current approach, when implemented on dedicated hardware devices such as FPGA, can approach the real-time testing speed of steel production. In order to assess the segmentation quality of our model, we sample 5K positive and 5K negative pixels from the test set. This produces an unbiased evaluation to measure the per-pixel error rate. Random guessing reaches only 50% error, while our SEGNN obtains 5.8%. Figure 6.2 shows a typical segmentation result for a test set image.

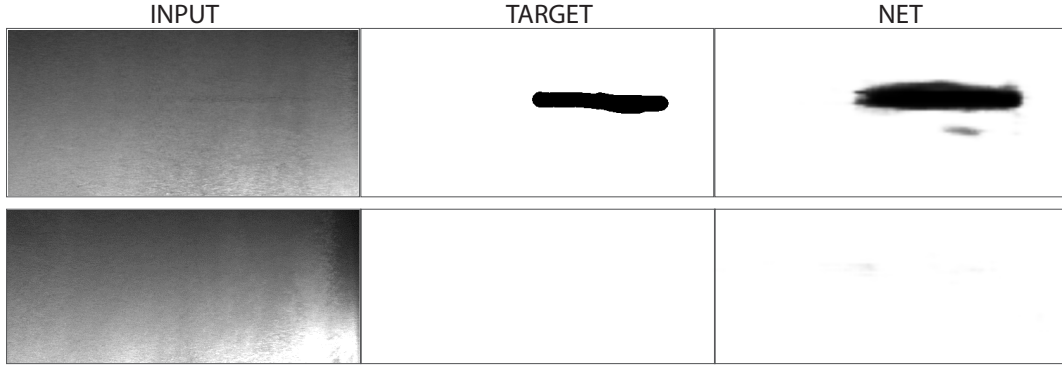


Figure 6.2. A typical steel defect example for the marker detection problem. The marker, whose location is shown in the target image, needs to be detected and a flag must be raised when the input image contains it. We can see that segmentation is almost perfect, illustrating the power of the proposed approach for industrial applications.

Table 6.2. Detection error results of our efficient learning framework for MPCNN. Test evaluation times for a given image are also reported along with the patch-based evaluation with the same implementation (e.g. Matlab).

Test err %	Patch (CPU)	Image (CPU)
2.3	110s	0.07s (1500x speed-up)

In order to perform detection, the segmentation output images need to be processed. This can involve quite long and elaborated processing stages. Our detection pipeline instead, thanks to a very good segmentation, is designed to be as simple as possible; it counts the number of pixels classified as defected. It works as follows. After learning a SEGNN on the training set, we determine on the validation set the threshold yielding the best detection performance. Such threshold is a single number above which each output pixel probability is converted into foreground.

A given image is flagged as containing a defect if the number of “defect” pixels it contains exceeds a given threshold. The threshold is set to 5000 (i.e. half the area of the smallest conceivable defect) and is not critical.

Table 6.2 shows detection performance. SEGNN makes only 3 mistakes and correctly detects 3 additional defects mislabeled during annotation.

### 6.1.3 Multiple detections per image

For this experiment we use a complete recording of a steel coil in the hot-strip mill section of the production line. The dataset contains gray-level images of  $2048 \times 4096$  pixels. Each of the images can contain several, possibly different, defects and it is not enough to raise a flag when the image contains an object of interest as in Section 6.1.2. A localization of the defected areas must be provided in form of ROIs which is a much harder problem than simply detecting whether there is an anomaly somewhere in the image as for Sec. 6.1.2 – which can be solved using a threshold on the number of pixels classified as defect for a given image because a ROI is not required.

We consider only the 104 images which contain at least one defect. As with almost all industrial data, there are images with incomplete annotation or images for which the anomalous regions are not detected. This makes learning difficult and trainable models barely generalize to unseen data due to overfitting. Of course, with conventional hand-crafted pipelines, where knowledge of the particular structure of the defect is plugged-in, this issue becomes less relevant. Finding an easy way to plug-in such knowledge, if any, is however hard, suboptimal and not always possible (e.g. see multi-variate data).

The model is trained on 70% of the images that contained at least one defect to speed-up learning, validated on 10% to find threshold parameters, and tested on the rest. The detection algorithm to post-process output probability maps proceeds as follows: apply closing with square of size 5, fill holes (e.g. a commonly used mathematical morphology operator) and computes connected components. Discards then components smaller than the given area threshold. All images are downsampled by a factor of 8 for ease of training and also because competitor detectors perform usually better with smaller resolutions.

The SEGNN is compared to two state-of-the-art detectors, HOG [Dalal and Triggs, 2005] and LBP [Ojala et al., 2002], based on the same window size of our network and linear SVM as region classifier as explained in Dalal and Triggs [2005], with the exception of the multi-scale non-maxima suppression which is not needed in steel industry (e.g. scale is a discriminant feature, scale invariance mostly disrupt performance). We used as many positive and negative samples as our 16GB system allows for training and testing on all sub-windows for any given image in a convolutional fashion. As a further remark on the on-line feasibility of our approach consider that an efficiently implemented HOG descriptor of vl-feat [Vedaldi and Fulkerson, 2008] to scan an entire image takes 12.2 seconds, while the SEGNN detector takes only 1.4 seconds and has three times better average-precision.

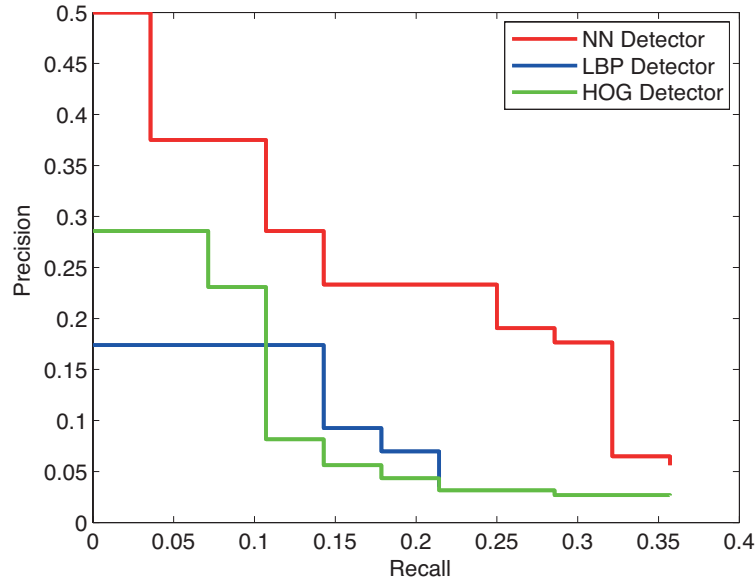


Figure 6.3. Precision-Recall curves for the three methods on the bobine dataset. We clearly see that SEGNN has a much better precision for any recall value. Of particular interest is the good performance when recall is high as it means that less false positive alarms are raised.

Table 6.3. Average Precision results for several models on the full steel coil detection problem. All methods work on the same support of  $32 \times 32$  pixels.

Model	#Input	AP
HOG Detector	1	3.93%
LBP Detector	1	3.06%
SEGNN Detector	1	9.53%

For this experiment, as multiple detections are to be produced, the well established Pascal VOC protocol was adopted. We report the average precision and plot the precision-recall curves in Table 6.3 and Fig. 6.3. Some exemplar detections for a qualitative evaluation are shown in Fig. 6.4.



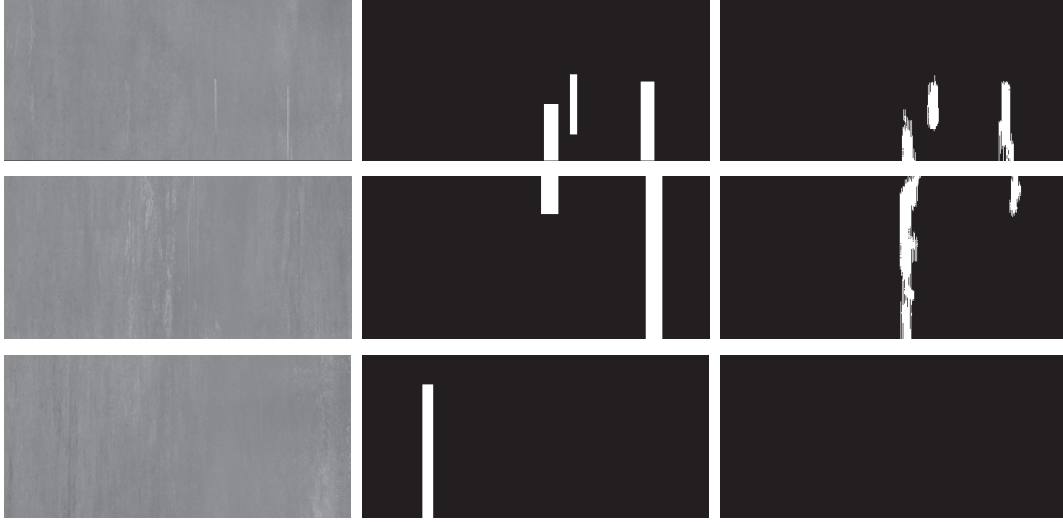


Figure 6.4. A typical steel defect example from the coil dataset. This is a more challenging problem than the one of Sec. 6.1.2. Given the images in the first column we are asked to produce the ROI as indicated in the second column, the ground-truth data. SEGNN results, for these test images, are reported in the third column. The top-most result is almost perfect. For the second image from the top, the long vertical stripe of SEGNN, even if larger than desired, after a visual inspection makes lot of sense as it closely capture the defected region.

## 6.2 Results on multi-variate images

An even more challenging area of application for detection systems is the one of multi-variate imaging. Here the input representation is no longer a gray or color image, but every pixel is now a high-dimensional vector.

This is a perfect test-bed with which to demonstrate the full potential of the system which only requires having a desired target to be extended to this type of image.

In this setting conventional feature extraction and segmentation schemes are not directly applicable because of all the underlying assumptions used to build them. For example, even the gradient of a multi-variate image is not clearly defined.

To evaluate the model on multi-variate images, a synthetic dataset was generated. It has 4 defects, each one with its own signature, plus the background as shown in Figure 6.5. Each signature is a template vector with 23 channels



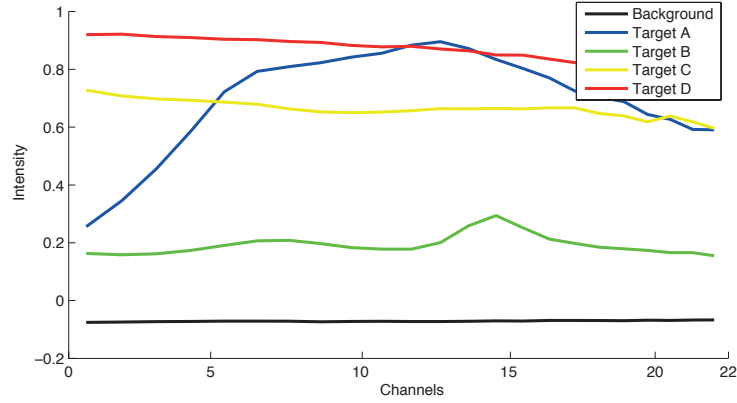


Figure 6.5. Signatures used for the multi-variate dataset generation. Note that target C and D are extremely difficult to discriminate, they have no relevant peaks or differences but belong indeed to two different defect classes.

and defines the prototypes used for the images. In a real scenario we would have to use, for example, the signature of iron oxide and the one of the clean metal as references. Three possible shapes are also defined for the defects, a circle, a vertical elongated and a texture on a variable size patch. To make the task non trivial white Gaussian noise is added for both the background and the blending of foreground and background. First the background is generated and after defects are generated and added to it, in a process which we call as mixing. A high mixing factor makes the task very hard as several signatures are linearly combined and noise is added. In all experiments we used a mixing factor of 0.7, where 1 means pure overlay, the easiest scenario.

The generated dataset is composed of 150 images, of which the last 50 are used for testing; a representative false colored image (e.g. we take three components out of 23 and show them as an RGB image) is shown in Fig. 6.6. SEGNN is evaluated on the two following problems:

- Task 1 Detection of any of the defects from the background. This is the easiest of the scenarios for which ambiguities between various targets do not play any crucial role.
- Task 2 Detection of a subset of the defects. The second problem shows the most interesting aspect of our approach. The two most problematic signatures, targets C and D, are considered one as defect and one as background. With such choice anomaly detection methods, and more generally all unsupervised methods are doomed to fail as they can only compute, or estimate,

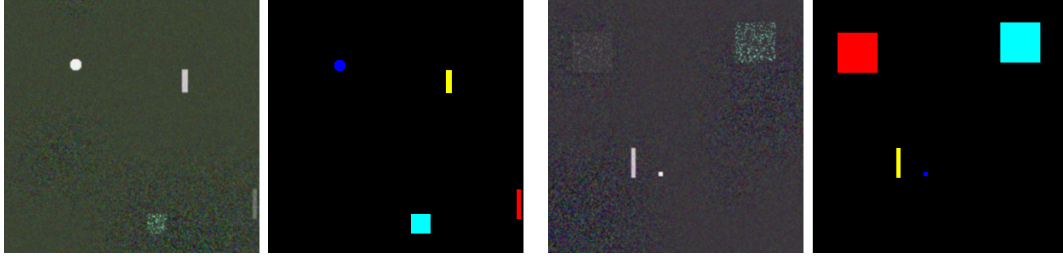


Figure 6.6. Example images from the multi-variate synthetic dataset of steel defects. Note that the data has 23 channels, but only channels 1, 10 and 23 are plotted in RGB for illustrative purpose only. Target images, second and fourth column from the left, show the defect and its corresponding color coded class label. Each defect, such as the red one for example, can come in different shapes so that learning the shape of the object does not suffice to obtain good performance. The first and third columns show the images which compose the synthetic dataset.

a metric to measure how the various targets differ from the known background. Clearly C and D are *far* from the background and there is no easy way to separate them without doing feature learning.

SEGNN is compared to two state-of-the-art image processing techniques; an unsupervised anomaly detector and a supervised target detector as described in the work of [Manolakis et al. \[2003\]](#) and [Kraut et al. \[2005\]](#), respectively. They are among the best methods to tackle the tasks of our experimental evaluation and have been long studied in the literature. A remark concerns the Target Detector. It uses the actual reference signatures and therefore has perfect knowledge on the task which needs to solve, in other words it has the reference model which SEGNN has to infer from the data. In a less controlled environment such signatures must be obtained via an additional algorithm such as VCA (vertex component analysis) [[Nascimento and Dias, 2005](#)] or NMF (non-negative matrix factorization) [[Lee and Seung, 2001](#)]. In fact, a least square problem needs to be solved at least, with non negativity constraints and latent representation which sums to 1 (that is, the ratio of each signature in the measurement). The non-negativity is a physical constraint as spectra cannot be negatives. The constraint to sum to 1 is to model that measured value are proportions of over-imposed spectra; a value of 1 means pure and lower values mean different combinations, or mixing.

Experiment are performed using as input to SEGNN the raw signals (23 chan-

Table 6.4. Average Precision results for several models on the multi-variate dataset. Every model has to detect any of the 4 possible defects out of the background.

Model	#Input	AP
Anomaly Detector	23	36.31 %
Target Detector	23	81.41 %
SEGNN Detector	6	92.48 %
SEGNN Detector	23	94.96 %

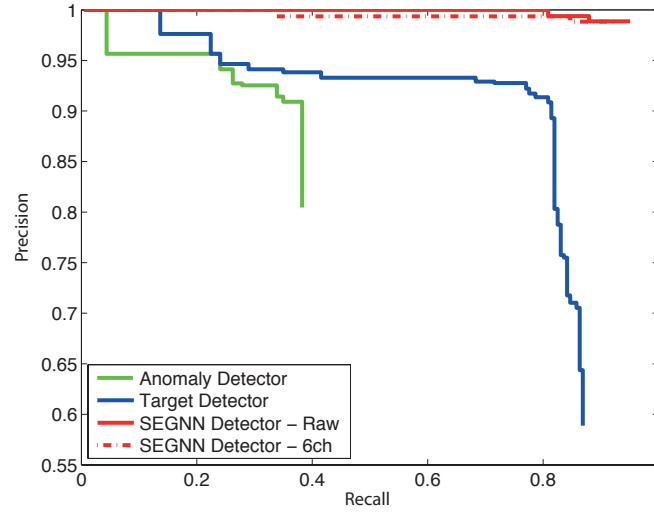


Figure 6.7. Precision-Recall curves for the three methods on the multi-variate dataset for the detection task where the SEGNN detector outperforms by a large margin its best competitor. Please note that SEGNN does not require any knowledge on the data whereas the Target Detector needs the signature of the defects. We implicitly solve the related inverse problem of finding such signatures through learning the segmentation map, in our opinion a remarkable achievement of our model.

nels) and the output of the anomaly and target detection together (6 channels). Competitor methods use the raw signal. The latter is to reduce the total number of computations and memory which can be beneficial in some applications where hardware has very limited power.

Table 6.5. Average Precision results for several models on the multi-variate dataset where Target D is considered as background. Please note that Target C and D are very similar and therefore discriminating between them makes this task extremely hard; the Target Detector has a consistent drop in performance whereas our deep learning approach suffers only a minor degradation.

Model	#Input	AP
Anomaly Detector	23	21.22 %
Target Detector	23	52.23 %
SEGNN Detector	6	77.11 %
SEGNN Detector	23	82.53 %

The same model was used in all the experiments of this section, simply changing the number of input channels from 6 to 23 and vice-versa, a remarkable feature of our supervised approach which does not require particular tuning or prior knowledge. Indeed the model is robust to architecture changes, performance does not drop if few filters are added or removed in each layer.

In Table 6.4 average precision results for Task 1 are reported and corresponding precision-recall curves are shown in Fig. 6.7. SEGNN outperforms its best competitor by a large margin without knowing any of the defect signatures.

Table 6.5 and Fig. 6.8 show the results for Task 2. SEGNN does not suffer the same performance degradation as the competitors because the feature extraction is learned so that the detector is tailored to the specific properties of the data.

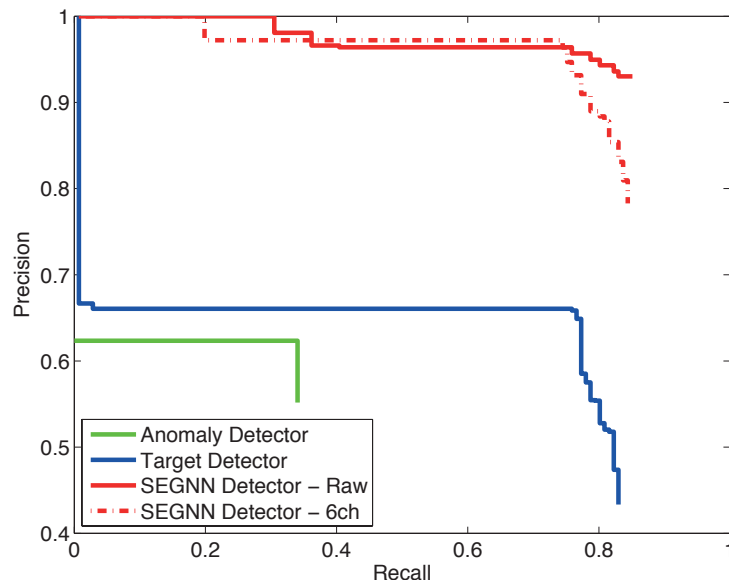


Figure 6.8. Precision-Recall curves for the three methods on the multi-variate dataset where class 4 is labeled as background. When the 23-channel raw signal is used SEGNN achieves almost perfect precision at very high recall, contrary to other approaches which sharply decay.



## Chapter 7

# Learning Morphological Operators using Counter-Harmonic Mean

Morphological operators are powerful nonlinear operators that, when properly designed, are crucial to cutting edge performance. Consequently, many attempts have been made to learn them aiming at even better performance.

Unfortunately, finding the proper pipeline of morphological operators and structuring elements for real applications is a cumbersome and time consuming task. In the machine learning community there has always been great deal of interest in learning such operators, but due to the non-differentiable nature of the max/min filtering, which limits what can be done with commonly used optimization frameworks, only a few approaches have been found to succeed, notably one based on LMS (gradient steepest descent algorithm) for rank filters formulated with the sign function by [Salembier \[1992a,b\]](#). This idea was later revisited by [Pessoa and Maragos \[1998\]](#) in a neural network framework combining morphological/rank filters and linear FIR filters. Other attempts from the evolutionary community (e.g., genetic algorithms [[Harvey and Marshall, 1996](#)] or simulated annealing [[Wilson, 1993](#)]) use black-box optimizers to circumvent the differentiability issue. However, most of the proposed approaches do not cover all operators. More importantly, they cannot learn both the structuring element and the operator, e.g., [[Nakashizuka et al., 2010](#)]. This is an important limitation as it makes it very hard or even impossible to compose complex filtering pipelines. Furthermore, such systems are usually limited to a very specific application, and generalize poorly to complex scenarios.

Inspired by recent work of [Angulo \[2010\]](#) on counter-harmonic mean asymptotic morphology, this chapter introduces a novel framework to learn pipelines of morphological operators. It combines concepts from convolutional neural net-

works with a new type of layer that permits complex pipelines through multiple layers and therefore extend the CNN class of models to Morphological Convolutional Neural Networks (MCNN).

In Section 7.1 the theoretical background on counter-harmonic mean asymptotic morphology is introduced. It builds the basis for the new key element of convnets, the PConv layer, detailed in Section 7.2. A thorough experimental evaluation follows. From increasing order of complexity we first show how the model learns dilation/erosion, then their composition into opening/closing and ultimately the widely used top-hat transform (i.e., residue of opening/closing). An important real-world application from the steel industry is presented and a sample application to denoising, where operator learning outperforms hand-crafted structuring elements, concludes the chapter. <sup>1</sup>

## 7.1 Asymptotic morphology using Counter-Harmonic Mean

Let us begin from the notion of counter-harmonic mean of Bullen [1987], initially used by van Vliet [2004] for constructing robust morphological-like operators. More recently, its morphological asymptotic behavior was characterized in the work of Angulo [2010].

Let  $f(x)$  be a 2D real-valued image, i.e.,  $f : \Omega \subset \mathbb{Z}^2 \rightarrow \mathbb{R}$ , where  $x \in \Omega$  denotes the coordinates of the pixel in the image domain  $\Omega$ . Given a (positive) weighting kernel  $w : W \rightarrow \mathbb{R}_+$ ,  $W$  being the support window of the filter, the *counter-harmonic mean (CHM) filter of order  $P$* ,  $-\infty \leq P \leq \infty$  is defined by,

$$\kappa_w^P(f)(x) = \frac{(f^{P+1} * w)(x)}{(f^P * w)(x)} = \frac{\int_{y \in W(x)} f^{P+1}(y) w(x-y) dy}{\int_{y \in W(x)} f^P(y) w(x-y) dy}, \quad (7.1)$$

where  $f^P$  is the image, each pixel value of  $f$  is raised to power  $P$ ,  $/$  indicates pixel-wise division, and  $W(y)$  is the support window of the filter  $w$  centered on point  $y$ . We note that the CHM filter can be interpreted as  *$P$ -deformed convolution*, i.e.,  $\kappa_w^P(f)(x) \equiv (f *_P w)(x)$ .

For  $P \gg 0$  ( $P \ll 0$ ) the pixels with largest (smallest) values in the local neighborhood  $W$  will dominate the result of the weighted sum (convolution), therefore morphological dilation and erosion are the limit cases of the CHM

<sup>1</sup>This chapter is based on Masci et al. [2013a]. A free implementation and a demo program to reproduce the results of the dilation/erosion experiments here reported is available at <http://www.idsia.ch/~masci/pconv>



filter, i.e.,

$$\lim_{P \rightarrow +\infty} (f *_P w)(x) = \sup_{y \in W(x)} f(y) = \delta_W(f)(x), \quad (7.2)$$

and

$$\lim_{P \rightarrow -\infty} (f *_P w)(x) = \inf_{y \in W(x)} f(y) = \varepsilon_W(f)(x), \quad (7.3)$$

where  $W$  plays the role of the structuring element.

As proven earlier by [Angulo \[2010\]](#), apart from the limit cases (e.g., a typical order of magnitude of  $5 \leq |P| < 10$ ), we have the following behavior:

$$(f *_P w)(x) |_{P \gg 0} \approx \sup_{y \in W(x)} \left\{ f(y) + \frac{1}{P} \log(w(x-y)) \right\}, \quad (7.4)$$

$$(f *_P w)(x) |_{P \ll 0} \approx \inf_{y \in W(x)} \left\{ f(y) - \frac{1}{P} \log(w(x-y)) \right\}, \quad (7.5)$$

which can be interpreted, respectively, as the nonflat dilation (supremal convolution) and nonflat erosion (infimal convolution) using the structuring function  $b(x) = \frac{1}{P} \log(w(x))$ . By using constant weight kernels, i.e.,  $w(x) = 1$  if  $x \in W$  and  $w(x) = 0$  if  $x \notin W$ , and  $|P| \gg 0$ , we just recover the corresponding flat structuring element  $W$ , associated to the structuring function  $\mathfrak{w}(x) = 0$  if  $x \in W$  and  $\mathfrak{w}(x) = -\infty$  if  $x \notin W$ .

From a precise morphological viewpoint, we notice that for finite  $P$  one cannot guarantee that  $(f *_P w)(x)$  yields exactly a pair of dilation/erosion, in the sense of commutation with max/min [[Serra, 1982](#); [Soille, 1999](#)]. Consequently, *stricto sensu*, we can only name them as pseudo-dilation ( $P \gg 0$ ) and pseudo-erosion ( $P \ll 0$ ). The asymptotic cases of the CHM filter can be also combined to approximate opening and closing operators, i.e.,

$$\begin{cases} ((f *_P w) *_P w)(x) \xrightarrow{P \gg 0} \gamma_W(f)(x), \\ ((f *_P w) *_P w)(x) \xrightarrow{P \ll 0} \varphi_W(f)(x). \end{cases} \quad (7.6)$$

## 7.2 Method

Thanks to the inspiring work of [Angulo \[2010\]](#), summarized in the previous section, we now introduce the novel morphological layer based on the CHM filter formulation, referred to as the **PConv** layer.

For a single channel image  $f(x)$  and a single filter  $w(x)$  the PConv layer performs the following operation

$$PConv(f; w, P)(x) = \frac{(f^{P+1} * w)(x)}{(f^P * w)(x)} = (f *_P w)(x) \quad (7.7)$$

PConv is parametrized by  $P$ , a scalar which controls the type of operation ( $P < 0$  pseudo-erosion,  $P > 0$  pseudo-dilation and  $P = 0$  standard linear convolution), and by the weighting kernel  $w(x)$ , where the corresponding asymptotic structuring function is given by  $\mathfrak{w}(x) = \log(w(x))$ . Since this formulation is differentiable we can use gradient descent on these parameters.

The gradient of the PConv layer is computed by backpropagation and is as follows:

$$\frac{\partial L}{\partial w} = \tilde{f}^{P+1} * \Delta_U + \tilde{f}^P * \Delta_D \quad (7.8)$$

$$\frac{\partial L}{\partial P} = f^{P+1} \circ \log(f) \circ \left( \frac{f}{f^P * w} * \tilde{w} \right) + f^P \circ \log(f) \circ (\Delta_D * \tilde{w}) \quad (7.9)$$

where

$$\Delta_U(x) = \frac{f(x)}{(f^P * w)(x)}; \quad \Delta_D(x) = \frac{-f(x) \circ (f^{P+1} * w)(x)}{(f^P * w)(x)}. \quad (7.10)$$

and  $\tilde{f}, \tilde{w}$  indicate flipping along the two dimensions and  $\circ$  indicates element-wise multiplication. The partial derivative of the PConv layer with respect to its input (to back-propagate the gradient) is instead

$$\frac{\partial (f *_P w)(x)}{\partial f} = \Delta_U(x) + \Delta_D(x). \quad (7.11)$$

Section 2.3.2 illustrated the importance of the top-hat transform for steel industry because of its ability to extract, preserving texture and border, a structure of interest from an image. Learning the top-hat operator requires though a *short-circuit* in the network to allow for subtracting the input image (or the result of an intermediate layer) from the output of a filtering chain. For this purpose we introduce the **AbsDiffLayer** which takes two layers as input and emits the absolute difference between them. Partial derivatives can still be back-propagated. Figure 7.1 shows a schematic representation of a network with a PConv layer, this is the architecture used for the dilation and erosion.

**Learning Algorithm.** Minimizing a PConv layer is a non-convex, highly non-linear operation prone to local convergence. Deep learning findings tell us that stochastic gradient descent is the most effective algorithm to train such complex models. In our experiments we use its full online version where weights are updated sample by sample. The learning rate decays during training. To further avoid bad local minima we use a momentum term. For the opening/closing tasks

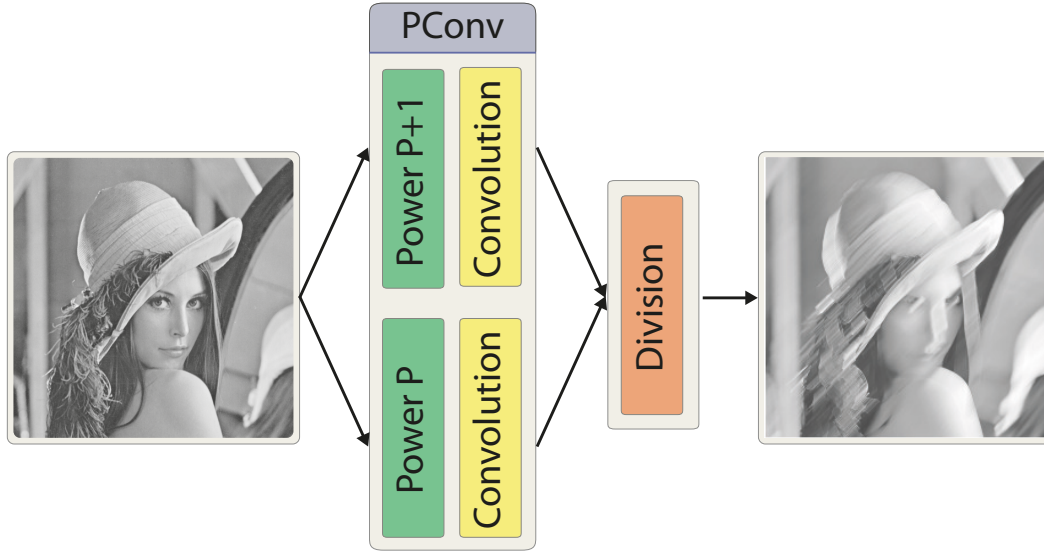


Figure 7.1. Illustrative example of a PConv layer. The network here represented can be used for learning dilation and erosion. In case of multiple convolutional kernels (i.e. multiple structuring elements in this setting) there will be a  $P$  for each one of them. This way it is possible to learn a richer set of operators.

we also alternate between learning  $P$ , keeping  $w$  fixed, and vice-versa. This is common in online dictionary learning and sparse coding. We also constrain  $w \geq 0$ .

## 7.3 Experiments

The MCNN was evaluated on several tasks. First, the quality of dilation/erosion operators is assessed in Section 7.3.1, this requires a single PConv layer and gives a good measure of how well training can be performed using the CHM derivation. Then, in Section 7.3.2, a two-layer network is investigated to learn openings/closings. This is already a challenging task hardly covered in previous approaches.

The experimental evaluation shows, in Section 7.3.3, that also the top-hat transform can be learned and for which a challenging steel industry application is reported (refer to Section 2.3.2 for further details on why this transform is important in steel industry). Using two learned morphological filters in each

PConv layer the MCNN learns to simultaneously detect two families of defects without resorting to multiple training stages. The implementation allows for learning multiple filters for every layer, thus producing a very rich set of filtered maps. Subsequent convolutional layers can learn highly nonlinear embeddings (we believe that this will also dramatically improve segmentation capabilities of such models). A simple CNN does *not* learn well pipelines of morphological operators. This is actually expected *a-priori* due to the nature of conventional convolutional layers, and shows the added value of the novel PConv layer.

As final benchmark the denoising problem is considered in Section 7.3.4. MCNN shows the superiority of operator learning over hand-crafting structuring elements for non-Gaussian (binomial and salt-and-pepper) noise. We also show that our approach performs well on total variation (TV) approximation for additive Gaussian noise.

In all our experiments we use stochastic gradient descent and a filter size of  $11 \times 11$  unless otherwise stated. The per-pixel mean-squared error loss (MSE) is used.

### 7.3.1 Learning dilation and erosion

In this first set of experiments the dilation and erosion operators are investigated. These form the basis of any morphological filtering and the ability to learn them puts solid foundations on the approach. The network, input with a clean image, must learn both the structuring element (i.e. convolutional kernel) and the type of operation (i.e. dilation or erosion) just from a target image which shows the desired transformation.

The dataset is created as follows: for every input image  $f_i$  of size  $512 \times 512$  we produce a target image  $t_j$  using a predetermined flat structuring element  $B_k$  and a predetermined operator:  $t_j = \delta_{B_k}(f_i)$  or  $t_j = \varepsilon_{B_k}(f_i)$ . We then train until convergence.

A CNN with equal topology (i.e. convolutional layer in place of the PConv layer) fails, producing mainly Gaussian blurred images, illustrating the need for a PConv layer to handle this kind of nonlinearity. A MCNN matches very closely, instead, the true underlying function.

Figure 7.2 shows the results of a dilation with three structuring elements: a line of 15 pixels oriented at  $45^\circ$ , a 5-pixel-sided square and a 5-pixel-sided diamond. Figure 7.3 shows similar results for the erosion transform.

Note that the learned weighted kernels  $w(x)$  are not exactly uniformly equal to 1. The corresponding morphological structuring function  $\mathfrak{w}(x)$ , obtained after applying the logarithm on the weights, produces a rather flat shape. In practice,

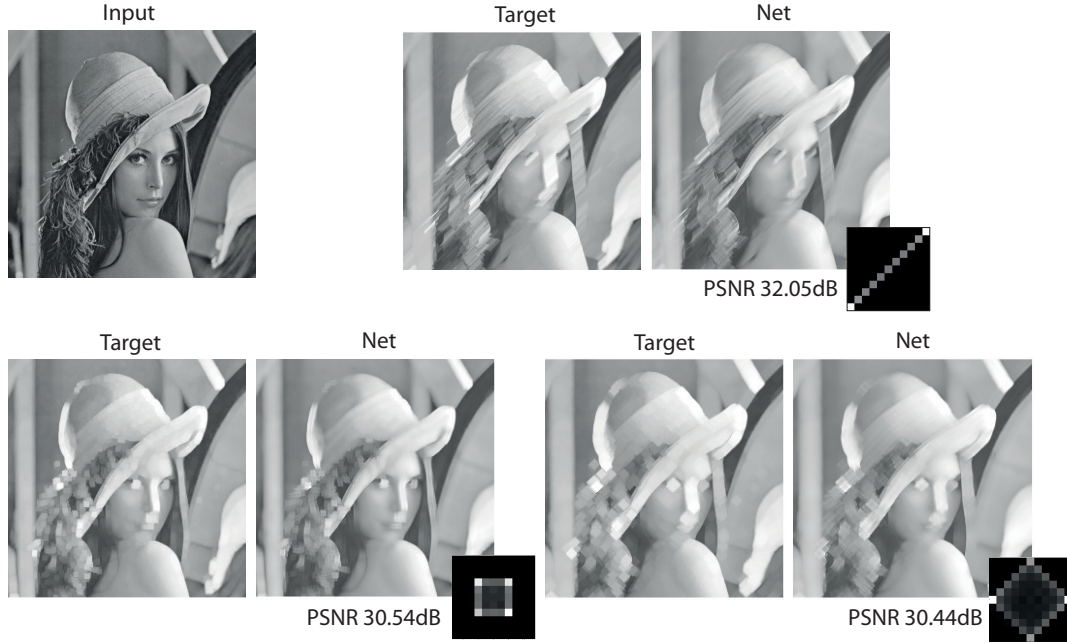


Figure 7.2. Examples of learning a dilation with three different structuring elements. The target and net output are slightly smaller than the original image due to valid convolution. The obtained kernel  $w(x)$  for each case is also depicted.

we observed that learning an erosion is slightly more difficult than learning the dual dilation. This is related to the asymmetric numerical behavior of CHM for  $P > 0$  and  $P < 0$ . Nevertheless, in all cases the learned operator has excellent performance.

### 7.3.2 Learning opening and closing

In this set of experiments we train the MCNN system to learn openings  $\gamma_{B_k}(f_i)$  and closings  $\varphi_{B_k}(f_i)$ . Learning such functions is extremely difficult because it involves learning simultaneously two PConv layers, one for the dilation and the other for the erosion, or vice-versa. To the best of our knowledge, this is the first work to do this in a flexible and gradient-based framework without any prior. For instance, in classical approaches, such as the one of [Salembier \[1992a\]](#) or in more recent ones as proposed by [Nakashizuka et al. \[2010\]](#), the operator needs to be fixed *a priori*.

Figure 7.4-(top row) shows an example of a closing with a line of length 10

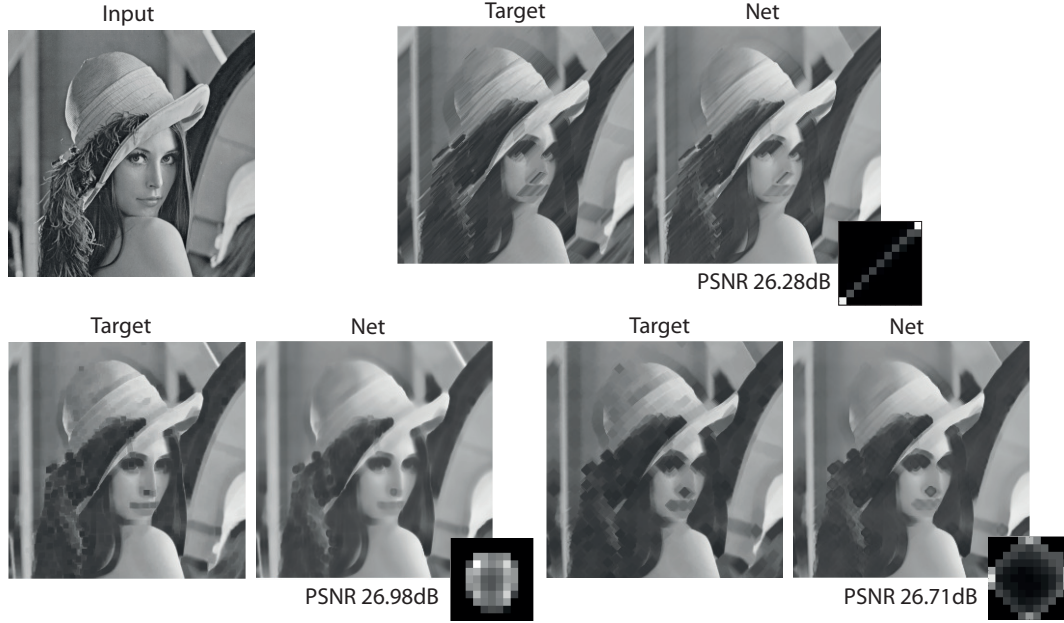


Figure 7.3. Examples of learning an erosion with three different structuring elements along with the learned kernel  $w(x)$ .

and an orientation of  $45^\circ$ , whereas Figure 7.4-(bottom row) shows an example of an opening with a square of size 5. In both cases, the obtained kernel for the first  $L1$  and second  $L2$  PConv layers are depicted. It is clear that the associated structuring element is learned with a good approximation. However, we also start to see that learning a flat opening/closing is remarkably hard and that the network output starts to be slightly “blurry”. This is due to the obtained values of  $P$ , e.g., in the closing  $P_{L1} = 6.80$  and  $P_{L2} = -8.85$ , in the opening  $P_{L1} = -7.64$  and  $P_{L2} = 7.07$  are in the interval of asymptotically unflat behavior. On the other hand, they are not totally symmetric. We intend to further study these issues in ongoing work.

### 7.3.3 Learning top-hat transform

The top-hat transform is particularly relevant in real applications such as the one of steel surface quality control. It is a powerful tool for defect detection as we showed in the test case scenario of Section 2.3.2.

Here we first show that our framework can learn such a transform, then, increasing the number of filters per layer from 1 to 2, we show that the MCNN





Figure 7.4. Top: an example of learning a closing operator where a line of length 10 and orientation  $45^\circ$  is used. Bottom: opening with a square structuring element of size 5. The network closely matches the output and almost perfectly learns the structuring elements in both PConv layers.

is also much more powerful jointly learning two transforms.

Having cast the learning to a neural network allows for easily constructing complex network topologies by linking several simple modules, or layers. We recall that the white top-hat is the residue of the opening, i.e.,  $\varrho_{B_k}^+(f_i) = f_i - \gamma_{B_k}(f_i)$ , and the black top-hat is the residue of the closing, i.e.,  $\varrho_{B_k}^-(f_i) = \varphi_{B_k}(f_i) - f_i$ . Thus, to learn a top-hat transform a **AbsDiffLayer** was introduced.

For this task a training set was generated by applying a white top-hat  $\varrho_B^+$ , where  $B$  is a disk of size 5 pixels. This operator extracts only the structures of size smaller than  $B$  and brighter than the background. Figure 7.5 shows the results for a single top-hat. It can be clearly seen that the network performs almost perfectly. To further assess the advantages of a PConv layer over a conventional convolutional layer, a CNN with identical topology was trained. The discrepancy between the two models, in terms of losses (MSE), is large:  $1.28\text{E-}3$  for MCNN and  $1.90\text{E-}3$  for the CNN. More parameters are required for a CNN to reach

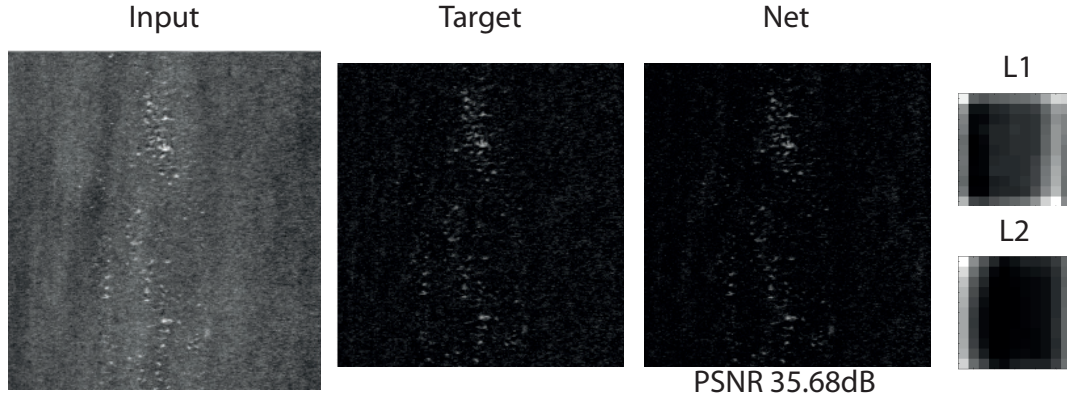


Figure 7.5. Learning a top-hat transform. The defected input image has bright spots to be detected. The network performs almost perfectly on this challenging task.

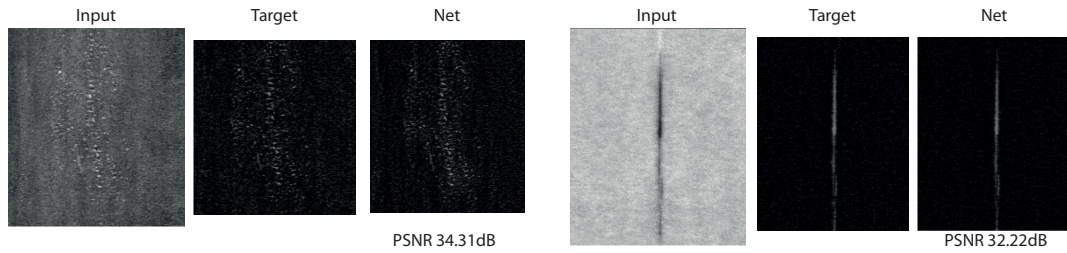


Figure 7.6. Learning two top-hat transforms. On the left, bright spots need to be detected. On the right, a dark vertical line. The network performs almost perfectly in this challenging task.

better performance, which is another advantage of the MCNN.

Figure 7.6 shows that by simply increasing the number of filters per layer two top-hat transforms can be simultaneously learned. A white top-hat with a disk of size 5 and a black top-hat with a line of size 10 and orientation of  $0^\circ$  are learned, and a convolutional layer is used to combine the output of the two operators. The architecture is as follows: 2 PConv layers, Conv layer, AbsDiff layer with the input. The network is almost perfect from our viewpoint. This opens up the possibility of using such a setup in more complex scenarios where several morphological operators should be combined. This is of great interest in multiple class steel defect detection.



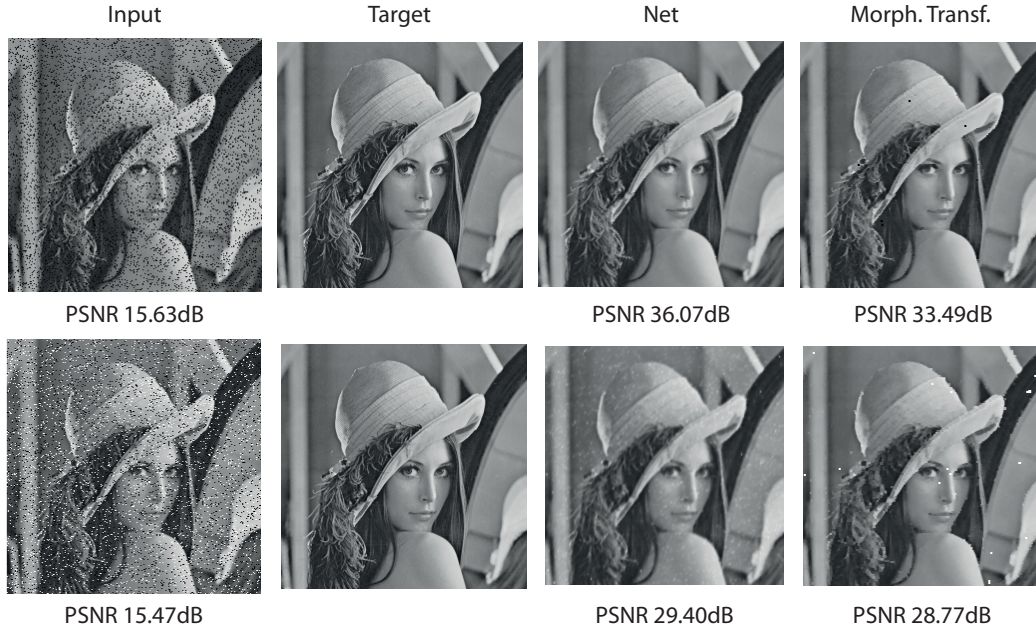


Figure 7.7. Top: Binomial noise removal task. The learned nonlinear operator performs better than the hand-crafted one by a relative margin of  $\approx 7.7\%$ . Learning uses noisy and original images—there is no prior on the task. Bottom: Salt’n’pepper noise removal task. Even here, the learned operator performs better than the corresponding morphological one.

#### 7.3.4 Learning denoising and image regularization

The MCNN is compared to conventional morphological pipelines in the denoising task. Morphological filters are recommended for non-Gaussian denoising. The purpose of this evaluation, however, is not to propose a novel noise removal approach, but to show the advantages of a learnable pipeline over a hand-crafted one.

We start with *binomial noise* where 10% of the image pixels are switched-off. The topology in this case is: 2 PConv layers and filter size of  $5 \times 5$ . This is compared to a closing with a square of size 2, empirically found to deliver best results. The task is made even harder by using larger-than-optimal support, keeping the filters of size  $11 \times 11$ . Training is performed in fully on-line fashion. While the target images are kept fixed, the input is generated by adding random noise sample by sample, so that the network never sees the same pattern twice. This creates a possibly infinite dataset with very small memory footprint. Fig-



Figure 7.8. Total Variation (TV) task. The network has to learn to approximate the TV output (target) by means of averaging two filtering pipelines.

ure 7.7-(top) compares the two approaches, and shows the noisy image. We see that learning substantially improves the PSNR measure.

An even more challenging task is 10% *salt’n’pepper* denoising, for which the network of 4 PConv layers (a very long pipeline) is compared to an opening with a square of size  $2 \times 2$  on a closing with the same structuring element. Training follows the same protocol as for the binomial noise, with images generated online. Figure 7.7-(bottom) shows results. Although we can observe some limitations of our approach, it still exhibits the best PSNR also in this application.

Finally the case of *total variation (TV)* restoration from an image corrupted by 6% additive Gaussian noise is considered. conventional morphological filtering does not excel at this task; only with the usage of connected filters better results can be obtained. A MCNN is trained to learn the mapping from noisy image to TV restored image. How well can it approximate any target transformation with a pseudo-morphological pipeline? The architecture is composed of 2 PConv layers with 2 filters each plus an averaging layer. Results are shown in Figure 7.8. Even in this case we are able to approximate with high accuracy an iterative process such as TV. Our network has, in fact, fixed complexity as its output is produced only after 4 steps, the convolutions which are required to traverse all layers in the MCNN.

## Chapter 8

# Multimodal Similarity-Preserving Hashing

Efficient computation of similarity between entries in large-scale databases has attracted increasing interest, given the explosive growth of data that has to be collected, processed, stored, and searched for. In particular, in the computer vision and pattern recognition community, this problem arises in applications such as image-based retrieval, ranking, classification, detection, tracking, and registration. In all these problems, given a query object (usually represented as a feature vector), one has to determine the closest entries (nearest neighbors) in a large database.

An even more challenging setting frequently arises in tasks involving multiple media or data coming from different modalities [Qi et al., 2011; Sharma et al., 2012]. For example, a medical image of the same organ can be obtained using different physical processes such as CT and MRI; a multimedia search engine may perform queries in a corpus consisting of audio, video, and textual information.

Since the notion of visual object similarity is rather elusive and cannot be measured explicitly, one often resorts to machine learning techniques that allow constructing a similarity measure from data examples. Such methods are generally referred to as *similarity* or *metric learning* and introduced in Section 2.4.

The extension of similarity learning to multimodal data has been addressed in the literature only very recently. Bronstein et al. [2010] proposed an extension of the SSH to the crossmodal setting, dubbed CM-SSH. McFee and Lanckriet [2011] proposed to learn multimodal similarity using ideas from multiple kernel learning [Bach et al., 2004; McFee and Lanckriet, 2009]. Multimodal kernel learning approaches have been proposed in Lee et al. [2009a] for medical image

registration. [Weston et al. \[2010\]](#) used multimodal embeddings for image annotation under the name of WSABIE (short for web-scale annotation by image embedding) for large-scale multi-modal information retrieval.

The appealing property of crossmodal similarity-preserving hashing methods like the CM-SSH of [Bronstein et al. \[2010\]](#) is the compactness of the representation and the low complexity involved in distance computation. However, CM-SSH is limited to linear projections which may not capture the structure of the data. Furthermore, it accounts only for the similarity *across* modalities, completely ignoring the data similarity *within* each modality. Finally, CM-SSH uses relaxation to solve the underlying optimization problem.

This chapter<sup>1</sup> proposes a novel multimodal similarity learning framework based on neural networks (NN hash) that tries to simultaneously learn two (or more) hashing functions that map the different modalities into a common binary space. We show experimentally that NN hash significantly outperforms state-of-the-art hashing approaches on multimedia retrieval tasks. The proposed approach has several advantages over the state-of-the-art. (1) it combines intra- and inter-modal similarity into a single framework: this allows richer information about the data to be exploited and can tolerate missing modalities. Several previous works can be considered as particular cases of our model. (2) it produces a compact binary code representation of the data, thus reducing storage and computational complexity of the similarity function that is better amenable for efficient indexing. (3) it solves the full optimization problem without resorting to relaxations as in SSH-like methods; it has been recently shown that such a relaxation degrades the hashing performance [[Masci et al., 2011b](#); [Strecha et al., 2012](#)]. (4) it introduces a novel coupled siamese neural network architecture to solve the optimization problem underlying the multimodal hashing framework. The use of neural networks can be very naturally generalized to more complex non-linear projections using multi-layered networks, thus allowing embeddings of arbitrarily high complexity following the spirit of deep learning.

Section 8.1 introduces the concepts of single- and multi- modal similarity preserving hashing and the current main algorithms to approach it. Section 8.2 explains the novel contribution of this chapter, the multimodal similarity preserving hashing framework based on the coupled siamese network. Finally Section 8.3 concludes with the experimental evaluation where the proposed approach is evaluated against the current state-of-the art for single- and multi-modal retrieval.

---

<sup>1</sup>This chapter is based on [Masci et al. \[2014a\]](#). A demo program to reproduce the results on CIFAR10 is available from the author's website.

## 8.1 Similarity-Preserving Hashing

Let  $X \subseteq \mathbb{R}^n$  and  $Y \subseteq \mathbb{R}^{n'}$  be two spaces representing data belonging to different modalities (e.g.,  $X$  are images and  $Y$  are text descriptions). Note that even though we assume that the data can be represented in the Euclidean space, the similarity of the data is not necessarily Euclidean and in general can be described by some metrics  $d_X : X \times X \rightarrow \mathbb{R}_+$  and  $d_Y : Y \times Y \rightarrow \mathbb{R}_+$ , to which we refer as *intra-modal dissimilarities*. Furthermore, we assume that there exists some *inter-modal dissimilarity*  $d_{XY} : X \times Y \rightarrow \mathbb{R}_+$  quantifying the “distance” between points in different modalities. To deal with these structures in a more convenient way, we try to represent them in a common metric space. In particular, the choice of the Hamming space offers significant advantages in the compact representation of the data as binary vectors and the efficient computation of their similarity.

**Unimodal similarity-preserving hashing** is the problem of representing data from one modality (say,  $X$ ) in the space  $\mathbb{H}^m = \{\pm 1\}^m$  of  $m$ -dimensional binary vectors with the Hamming metric  $d_{\mathbb{H}^m}(a, b) = \frac{m}{2} - \frac{1}{2} \sum_{i=1}^m a_i b_i$  by means of an embedding,  $\xi : X \rightarrow \mathbb{H}^m$  mapping similar points as close as possible to each other and dissimilar points as distant as possible from each other, such that  $d_{\mathbb{H}^m} \circ (\xi \times \xi) \approx d_X$ .

**Multimodal similarity-preserving hashing** is an extension of the former problem, in which two different modalities  $X, Y$  are represented in the common space  $\mathbb{H}^m$  by means of two embeddings,  $\xi : X \rightarrow \mathbb{H}^m$  and  $\eta : Y \rightarrow \mathbb{H}^m$  mapping similar points as close as possible to each other and dissimilar points as distant as possible from each other, such that  $d_{\mathbb{H}^m} \circ (\xi \times \xi) \approx d_X$ ,  $d_{\mathbb{H}^m} \circ (\eta \times \eta) \approx d_Y$ , and  $d_{\mathbb{H}^m} \circ (\xi \times \eta) \approx d_{XY}$ . In a sense, the embeddings act as a *metric coupling*, trying to construct a single metric that preserves both the intra- and inter-modal similarities. A simplified setting of the multimodal hashing problem used in [Bronstein et al. \[2010\]](#) is *cross-modality similarity-preserving hashing*, in which only the inter-modal dissimilarity  $d_{XY}$  is taken into consideration and  $d_X, d_Y$  are ignored.

In the rest of this chapter, we assume binary dissimilarities  $d_X, d_Y, d_{XY} \in \{0, 1\}$ , i.e., a pair of points can be either similar or dissimilar. This dissimilarity is usually unknown and hard to model, however, it should be possible to sample  $d_X, d_Y, d_{XY}$  on some subset of the data  $X' \subset X, Y' \subset Y$ . This sample can be represented as sets of similar pairs of points (*positives*)

$$\begin{aligned} \mathcal{P}_X &= \{(x \in X', x' \in X') : d_X(x, x') = 0\} \\ \mathcal{P}_Y &= \{(y \in Y', y' \in Y') : d_Y(y, y') = 0\} \\ \mathcal{P}_{XY} &= \{(x \in X', y \in Y') : d_{XY}(x, y) = 0\}, \end{aligned}$$

and similarly defined sets  $\mathcal{N}_X, \mathcal{N}_Y$ , and  $\mathcal{N}_{XY}$  of dissimilar pairs of points (*neg-*

atives). In many practical applications such as image annotation or text-based image search, it might be hard to get the inter-modal positive and negative pairs, but easy to get the intra-modal ones.

The problem of multimodal similarity-preserving hashing boils down in finding two embeddings  $\xi : X \rightarrow \mathbb{H}^m$  and  $\eta : Y \rightarrow \mathbb{H}^m$  minimizing the aggregate of false positive and false negative rates,

$$\begin{aligned} \min_{\xi, \eta} \quad & \mathbb{E}\{d_{\mathbb{H}^m} \circ (\xi \times \xi) | \mathcal{P}_X\} + \mathbb{E}\{d_{\mathbb{H}^m} \circ (\eta \times \eta) | \mathcal{P}_Y\} - \\ & \mathbb{E}\{d_{\mathbb{H}^m} \circ (\xi \times \xi) | \mathcal{N}_X\} - \mathbb{E}\{d_{\mathbb{H}^m} \circ (\eta \times \eta) | \mathcal{N}_Y\} + \\ & \mathbb{E}\{d_{\mathbb{H}^m} \circ (\xi \times \eta) | \mathcal{P}_{XY}\} - \\ & \mathbb{E}\{d_{\mathbb{H}^m} \circ (\xi \times \eta) | \mathcal{N}_{XY}\}. \end{aligned} \quad (8.1)$$

In what follows, we briefly review the existing approaches to supervised similarity-preserving hashing.

### 8.1.1 Supervised single-modality similarity-preserving hashing

In his Ph.D. dissertation, [Shakhnarovich et al. \[2003\]](#) introduced one of the first supervised hashing techniques called similarity-preserving hashing (SSH). The author proposed to regard the construction of an LSH-like similarity-preserving hash as a binary classification problem, in which pairs of points  $(\mathbf{x}, \mathbf{x}')$  are assigned positive or negative labels. The minimization of the expected Hamming distance,  $d_{\mathbb{H}^m}$ , on the set of positive pairs (and, respectively, its maximization on the negative set) can be achieved by minimizing the exponential loss of the form

$$\mathbb{E} \left\{ \exp(-\ell \xi(\mathbf{x})^T \xi(\mathbf{x}')) \right\} = \mathbb{E} \left\{ \prod_{i=1}^M \exp(-\ell \xi_i(\mathbf{x}) \xi_i(\mathbf{x}')) \right\},$$

where  $\ell = +1$  for a positive pair, and  $\ell = -1$  for a negative one indicates the ground-truth similarity. Observing the separability of the exponential loss, the author proposed to train the individual bits  $\xi_i$  of the embedding sequentially as *weak learners* using standard boosting techniques. In particular, Shakhnarovich considered linear embeddings of the form:

$$\mathbf{x}_i(\mathbf{x}) = \text{sign}(\mathbf{e}_{k_i}^T \mathbf{x} + a_i),$$

where  $\mathbf{e}_{k_i}$  is a standard basis vector acting as a feature selector, and  $a_i$  is a threshold.

The sequential construction of binary codes is clearly suboptimal. As a result, SSH typically requires relatively long codes to achieve good performance.



A remedy to this problem was proposed in the DiffHash scheme introduced by [Strecha et al. \[2012\]](#). The authors considered linear embeddings of the form  $\xi(\mathbf{x}) = \text{sign}(\mathbf{P}\mathbf{x} + \mathbf{a})$  trained by minimizing a quadratic loss

$$\mathbb{E} \left\{ \|\xi(\mathbf{x}) - \xi(\mathbf{x}')\|_2^2 | \mathcal{D} \right\} - \alpha \mathbb{E} \left\{ \|\xi(\mathbf{x}) - \xi(\mathbf{x}')\|_2^2 | \mathcal{N} \right\}, \quad (8.2)$$

with the parameter  $\alpha$  controlling the relative importance of false positives and negatives. By relaxing the problem through the removal of the sign function,  $\mathbf{P}$  can be found as the  $m$  smallest negative eigenvectors of the difference of the covariance matrices  $\mathbf{C}_{\mathcal{D}} - \alpha \mathbf{C}_{\mathcal{N}}$ , with  $\mathbf{C}_{\mathcal{D}} = \mathbb{E}\{(\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^T | \mathcal{D}\}$  and  $\mathbf{C}_{\mathcal{N}}$  defined likely on the negative pairs. Once the projection matrix  $\mathbf{P}$  has been found, the thresholds  $\mathbf{a}$  are found by solving  $m$  independent one-dimensional minimization problems. The authors showed that a globally optimal  $a_i$  can be computed from the cumulative histograms of  $\mathbf{p}_i^T \mathbf{x}$ .

Despite its simplicity and computational efficiency, the main drawback of DiffHash is the fact that it is limited to linear projections, which might not be able to properly capture the intricate structure of the data. In machine learning, it is common to introduce non-linearity into linear projection-based schemes via the kernel trick. Generalizing kernelized LSH [[Kulis and Grauman, 2009](#)] to the supervised setting, [Liu et al. \[2012\]](#) proposed the kernelized supervised hashing (KSH) scheme in which they considered embeddings of the form  $\xi(\mathbf{x}) = \text{sign}(\mathbf{P}\mathbf{k}(\mathbf{x}))$ , with  $\mathbf{P}$  being an  $m \times r$  projection matrix, and  $\mathbf{k}(\mathbf{x}) = (\kappa(\mathbf{x}, \mathbf{x}_1) - \mu_1, \dots, \kappa(\mathbf{x}, \mathbf{x}_r) - \mu_r)^T$  a non-linear map created by computing the inner product between  $\mathbf{x}$  and a fixed set of  $r$  points  $\mathbf{x}_1, \dots, \mathbf{x}_r$  drawn at random from the training set. The inner products are computed via the kernel function  $\kappa$ , which has to satisfy the standard Mercer conditions, and  $\mu_i$  is pre-computed as  $\kappa(\mathbf{x}, \mathbf{x}_i)$  averages over all  $\mathbf{x}$ 's in the training set. In this formulation, the supervised learning of the hash function boils down to minimizing a loss of the form

$$\mathbb{E} \left\{ \left( \frac{1}{m} \xi(\mathbf{x})^T \xi(\mathbf{x}') - \ell \right)^2 \right\}, \quad (8.3)$$

where  $\ell = +1$  or  $-1$  on  $(\mathbf{x}, \mathbf{x}')$  depending on whether it belongs to  $\mathcal{D}$  or  $\mathcal{N}$ , respectively. The authors show that the learning of  $\mathbf{P}$  can be performed either via greedy optimization similar to SSH, or by dropping the sign function and resorting to a spectral relaxation closely resembling DiffHash. In fact, depending on the choice of the optimization algorithm, KSH can be viewed as a kernelized version of either SSH or DiffHash. The greedy approximation or the spectral relaxation can be further refined by solving the highly non-convex problem

minimizing (8.3), in which the sign function is replaced by a smooth sigmoid approximation.

### 8.1.2 Supervised cross-modality similarity-preserving hashing

To the best of our knowledge, only one attempt has been made to date to generalize supervised hashing techniques to multiple modalities. Bronstein et al. [2010] studied the particular case of cross-modal similarity-sensitive hashing (without incorporating intra-modality similarity), with linear embeddings of the form  $\xi(\mathbf{x}) = \text{sign}(\mathbf{P}\mathbf{x} + \mathbf{a})$  and  $\eta(\mathbf{y}) = \text{sign}(\mathbf{Q}\mathbf{y} + \mathbf{b})$ , which can be considered an extension of SSH. The CM-SSH algorithm constructs the dimensions of  $\xi$  and  $\eta$  one-by-one using boosting. At each iteration, one-dimensional embeddings  $\xi_i(\mathbf{x}) = \text{sign}(\mathbf{p}_i^T \mathbf{x} + a_i)$  and  $\eta_i(\mathbf{y}) = \text{sign}(\mathbf{q}_i^T \mathbf{y} + b_i)$  are found using a two-stage scheme: first, the embeddings are linearized as  $\xi_i(\mathbf{x}) \approx \mathbf{p}_i^T \mathbf{x}$  and  $\eta_i(\mathbf{y}) \approx \mathbf{q}_i^T \mathbf{y}$  and the resulting objective is minimized to find the projection

$$\min_{\mathbf{p}_i, \mathbf{q}_i} \mathbb{E}\{\mathbf{x}^T \mathbf{p}_i \mathbf{q}_i^T \mathbf{y} | \mathcal{P}_{XY}\} - \mathbb{E}\{\mathbf{x}^T \mathbf{p}_i \mathbf{q}_i^T \mathbf{y} | \mathcal{N}_{XY}\}, \quad (8.4)$$

(here  $\mathbf{p}_i^T$  and  $\mathbf{q}_i^T$  are unit vectors representing the  $i$ th row of the matrices  $\mathbf{P}$  and  $\mathbf{Q}$ , respectively, and the expectations are weighted by per-sample weights adjusted by the boosting). With such an approximation, the optimal projection directions  $\mathbf{p}$  and  $\mathbf{q}$  have a closed-form expressions using the SVD of the positive and negative covariance matrices. At the second stage, the thresholds  $a_i$  and  $b_i$  are found by two-dimensional search.

SSH-type approaches (and consequently, CM-SSH) have several drawbacks. First, CM-SSH solves a particular setting of problem (8.1) with  $\mathcal{P}_{XY}, \mathcal{N}_{XY}$  only, thus ignoring the intra-modality similarities. Second, the assumption of separability (treating each dimension separately) and the linearization of the objective replace the original problem with a relaxed version, whose optimization produces suboptimal solutions. Finally, this approximation is limited to a relatively narrow class of linear embeddings that often do not capture well the structure of the data.

## 8.2 Multimodal NN hashing

Our approach for multimodal hashing is related to supervised methods for dimensionality reduction and in particular extends the framework of Bromley et al. [1994]; Hadsell et al. [2006]; Schmidhuber and Prelinger [1993]; Taylor et al.



[2011], also known as the *siamese architecture*. These methods learn a mapping onto a usually low-dimensional feature space such that similar observations are mapped to nearby points in the new manifold and dissimilar observations are pulled apart. In the simplest setting, the linear embedding  $\xi = \text{sign}(\mathbf{P}\mathbf{x} + \mathbf{a})$  is realized as a neural network with a single layer (where  $\mathbf{P}$  represents the linear weights and  $\mathbf{a}$  is the bias) and a sign activation function (in practice, we use a smooth approximation  $\text{sign}(x) \approx \tanh(\beta x)$ ). The parameters of the intra-modal embedding can be learned using back-propagation minimizing the loss

$$\begin{aligned} L_X &= \frac{1}{2} \sum_{(\mathbf{x}, \mathbf{x}') \in \mathcal{P}_X} \|\xi(\mathbf{x}) - \xi(\mathbf{x}')\|_2^2 \\ &+ \frac{1}{2} \sum_{(\mathbf{x}, \mathbf{x}') \in \mathcal{N}_X} \max\{0, m_X - \|\xi(\mathbf{x}) - \xi(\mathbf{x}')\|_2\}^2 \end{aligned} \quad (8.5)$$

w.r.t. the network parameters  $(\mathbf{P}, \mathbf{a})$ . In the same way, embedding  $\eta$  is learned by minimizing the loss  $L_Y$  w.r.t. parameters  $(\mathbf{Q}, \mathbf{b})$ . Note that for binary vectors (when  $\beta = \infty$ ), the squared Euclidean distance in (8.5) is equivalent, up to constants, to the Hamming distance. The second term in (8.5) is a *hinge-loss* providing robustness to outliers and produces a mapping for which negatives are pulled at  $m_X$  distance apart. The system is fed with pairs of samples for which a corresponding dissimilarity is known, 0 for positives and 1 for negatives (thus the name *siamese network*, i.e. two inputs and a common output vector). This approach has been also successfully applied by Taylor et al. [2011] to problems such as matching people in similar pose and which exhibits invariance to identity, clothing, background, lighting, shift and scale.

### 8.2.1 Coupled siamese architecture

In the multimodal setting, we have two embeddings  $\xi$  and  $\eta$ , each cast as a siamese network with parameters  $(\mathbf{P}, \mathbf{a})$  and  $(\mathbf{Q}, \mathbf{b})$ , respectively. Such an architecture allows similarity-sensitive hashing to be learned for each modality independently by minimizing the loss functions  $L_X, L_Y$ . In order to incorporate inter-modal similarity, we couple the two siamese networks by the cross-modal loss

$$\begin{aligned} L_{XY} &= \frac{1}{2} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{P}_{XY}} \|\xi(\mathbf{x}) - \eta(\mathbf{y})\|_2^2 \\ &+ \frac{1}{2} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{N}_{XY}} \max\{0, m_{XY} - \|\xi(\mathbf{x}) - \eta(\mathbf{y})\|_2\}^2, \end{aligned} \quad (8.6)$$

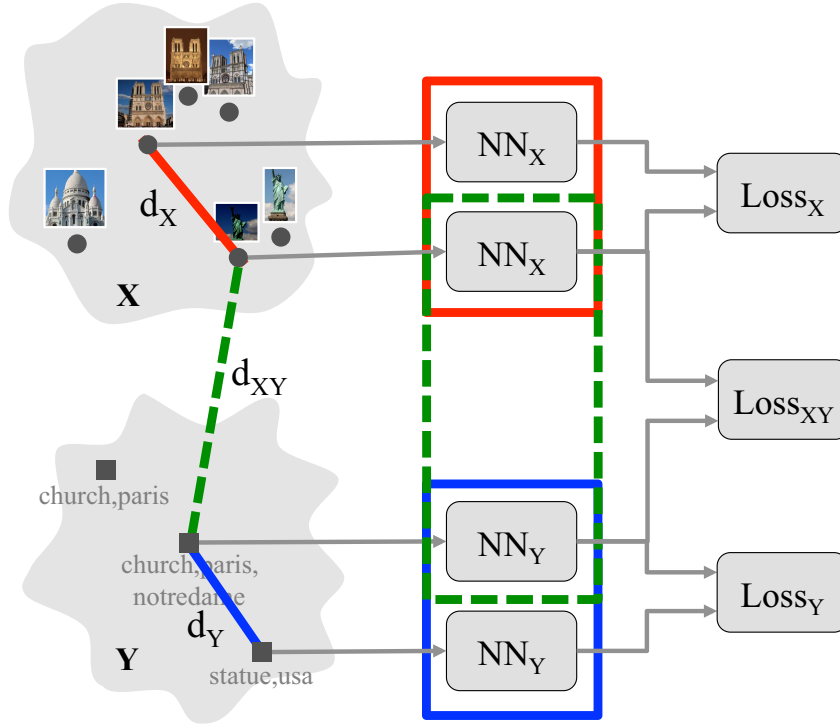


Figure 8.1. Schematic representation of the coupled siamese network. There are two networks, one for each of the modalities, coupled by the loss function  $L_{xy}$ .

thus jointly learning two sets of parameters for each modality. We refer to this model, which generalizes the siamese framework, as the *coupled siamese networks* (Figure 8.1).

Our implementation differs from the architecture of [Hadsell et al. \[2006\]](#) in the choice of the output activation function (we use tanh activation that encourages binary representations rather than a linear output layer). This way the maximum distance is bounded by  $\sqrt{4m}$ , and by simply enlarging the margin between dissimilar pairs we enforce the learning of codes which differ by the sign of their components; the Euclidean metric commutes with the Hamming metric in such case. Once the model is learned, hashes are produced by thresholding the output at zero.

This architecture can be extended to arbitrarily complex mappings by adding multiple layers of non-linearities. This has the advantage of scaling linearly

with the number of activations which is a very desirable property in large scale problems.

### Training

The coupled siamese network is trained by minimizing

$$\min_{\mathbf{P}, \mathbf{a}, \mathbf{Q}, \mathbf{b}} L_{XY} + \alpha_X L_X + \alpha_Y L_Y, \quad (8.7)$$

where  $\alpha_X, \alpha_Y$  are weights determining the relative importance of each modality. The loss (8.7), can be considered as a generalization of the loss in (8.1), which is obtained by setting  $\alpha_X = \alpha_Y = 1$ , margins = 0, and  $\beta = \infty$ . We call the setting  $\alpha_X, \alpha_Y > 0$  MM-NN (multimodal). Furthermore, for  $\alpha_X = \alpha_Y = 0$ , we obtain the particular setting of cross-modal loss (referred to in the following as CM-NN), whose relaxed version is minimized by the CM-SSH algorithm of [Bronstein et al. \[2010\]](#). It is also worth repeating that in many practical cases, it is very hard to obtain reliable cross-modal training samples ( $\mathcal{P}_{XY}, \mathcal{N}_{XY}$ ) but much easier to obtain intra-modal samples ( $\mathcal{P}_X, \mathcal{N}_X, \mathcal{P}_Y, \mathcal{N}_Y$ ). In the full multimodal setting ( $\alpha_X, \alpha_Y > 0$ ), the terms  $L_X, L_Y$  can be considered as a *regularization*, preventing the algorithm from over fitting.

We apply back-propagation to get the gradient of our model w.r.t. the embedding parameters. The gradient of the intra-modal loss function w.r.t. to the parameters of  $\xi$  is given by

$$\nabla L_X = \begin{cases} (\xi(\mathbf{x}) - \xi(\mathbf{x}'))(\nabla \xi(\mathbf{x}) - \nabla \xi(\mathbf{x}')) & (\mathbf{x}, \mathbf{x}') \in \mathcal{P}_X \\ (\xi(\mathbf{x}) - \xi(\mathbf{x}') - m_X)(\nabla \xi(\mathbf{x}) - \nabla \xi(\mathbf{x}')) & (\mathbf{x}, \mathbf{x}') \in \mathcal{N}_X, \\ & \text{and} \\ & m_X > \|\xi(\mathbf{x}) - \xi(\mathbf{x}')\|_2 \\ 0 & \text{else} \end{cases} \quad (8.8)$$

Equivalent derivation is done for the parameters of  $\eta$ . The gradient of the inter-modal loss function w.r.t. to the parameters of  $\xi$  is given by

$$\nabla L_{XY} = \begin{cases} (\xi(\mathbf{x}) - \eta(\mathbf{y}))\nabla \xi(\mathbf{x}) & (\mathbf{x}, \mathbf{y}) \in \mathcal{P}_{XY} \\ (\xi(\mathbf{x}) - \eta(\mathbf{y}) - m_{XY})\nabla \xi(\mathbf{x}) & (\mathbf{x}, \mathbf{y}) \in \mathcal{N}_{XY} \text{ and} \\ & m_{XY} > \|\xi(\mathbf{x}) - \eta(\mathbf{y})\|_2 \\ 0 & \text{else} \end{cases} \quad (8.9)$$

The model can be easily learned jointly using any gradient-based technique such as conjugate gradient or stochastic gradient descent. The latter is the preferred choice for large datasets as it has minimal memory footprint and performs many more updates of the parameters, one per sample in the fully online setting, speeding up convergence of deep architectures.

## 8.3 Experiments

In this section, NN hash is evaluated on several standard multimedia datasets: CIFAR10 [Krizhevsky, 2009], NUS [Chua et al., 2009], and Wiki [Rasiwasia et al., 2010] (see Table 8.1). All datasets were centered and unit-length normalized.

To avoid bad local minima a long list of techniques has been proposed, see Bengio [2009] and Section 2.2.4 for an overview. For this work the hybrid batch on-line approach of Ngiam et al. [2011] worked the best. Batches are sampled and the model is trained for only 5 iterations using L-BFGS, repeating until convergence. Because all parameters are learned, setting  $\beta = 1$  and adjusting the margin dependent on the code length delivered the best results.

In the experiments, a distinction is made between uni- and multi-modal training, where in the former the hash functions are learned on each modality individually without using the other modality, and in the latter, inter-modal information is also used. Likewise a distinction is made between uni- and cross-modal retrieval. In the former case, both the query and the database are from the same modality; in the latter case, the query and the database belong to different modalities.

In the unimodal setting, the following state-of-the-art hashing methods were compared: DiffHash [Strecha et al., 2012], SSH [Shakhnarovich et al., 2003], AGH [Liu et al., 2011], and KSH [Liu et al., 2012], using the code provided by the respective authors. In the cross-modal setting, we used Euclidean embedding by means of canonical correlation analysis (CCA) as a baseline, and compare to CM-SSH [Bronstein et al., 2010]. As a ‘sanity check’, hash functions trained in the multimodal setting were also tested on unimodal retrieval tasks. Ideally, the use of another modality information during training should improve (or at least not deteriorate) the performance of unimodal retrieval.

NN hash was tested in single-layer (L1) and two-layered (L2) configurations. We also distinguish between a version trained on inter-modal data only (CM-NN, corresponding to  $\alpha_X = \alpha_Y = 0$ ) and full multimodal version (MM-NN, using  $\alpha_X = \alpha_Y = 0.5$ ) making use of inter- and intra-modal training data. The archi-

Table 8.1. Summary of the experiments and datasets.

Dataset	Modalities		Classes	Testing	
	$n$	$n'$		queries	database
Wiki	128	10	10	693	2173
CIFAR10	384	486	10	1000	59000
NUS	500	1000	81	2100	193739

tecture of CM-NN L1 is directly comparable to CM-SSH.

We adopted the following rule of thumb for the margins: 3 for 12bit, 5 for 24 and 48 bit, 7 for 64 and 16 for 256bit.

NN hash, as it allows stochastic optimization, it does not run into memory problems when the number of data points grows. In fact we are not bounded at all by the size of the training set which is generated on the fly. This is a crucial difference between NN hash and other hashing approaches, since real-world datasets are typically orders of magnitude larger than what can be handled by standard batch methods.

The hash functions learned by each of the methods were applied to the data in the datasets, and the exact Hamming distance was used to rank the matches. Retrieval performance was evaluated using *mean average precision*  $mAP = \sum_{r=1}^R P(r) \cdot rel(r)$ , where  $rel(r)$  is the relevance of a given rank (one if relevant and zero otherwise),  $R$  is the number of retrieved results, and  $P(r)$  is *precision at  $r$* , defined as the percentage of relevant results in the first  $r$  top-ranked retrieved matches.

### 8.3.1 CIFAR10

CIFAR10 [Krizhevsky, 2009] is a set of 60K labeled images belonging to 10 different classes, sampled from the 80M tiny image benchmark of Torralba et al. [2008a]. The images are represented using 384-dimensional GIST and 486-dimensional HOG descriptors, used as two different modalities. Following Liu et al. [2012], we used a training set of 200 images for each class; for testing, we used a disjoint query set of 100 images per class and the remaining 59K images as database.

Table 8.2 shows the unimodal (GIST–GIST and HOG–HOG) retrieval performance; examples of a few top matches produced by different hashing algorithms are shown in Figure 8.2. We can see that NN-based methods significantly out-

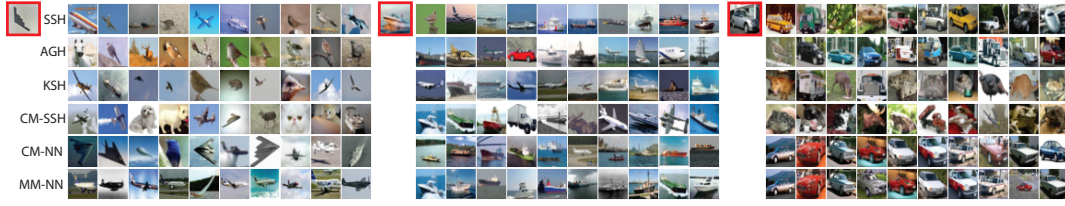


Figure 8.2. Unimodal retrieval on CIFAR dataset. Shown are top 10 matches to three different queries (marked in red) using different hashing method with codes of length 48. All NN methods are used in L2 configuration. CM-SSH, CM-NN and MM-NN were trained on multiple modalities, and used in this experiment for single modality retrieval.

perform all the rest of the methods, including the previous state-of-the-art AGH and KSH. Further significant improvement is achieved by using a two-layer configuration with 48 tanh units (NN-L2).

Table 8.2. Unimodal training and retrieval experiment on the CIFAR10 dataset. NN hash was trained on single modality only. Performance is shown as mAP in %.

GIST – GIST				HOG – HOG			
Method / $m$	12	24	48	12	24	48	
DiffHash	14.72	13.35	12.85	13.05	11.92	11.47	
SSH	15.42	16.75	17.06	15.49	16.15	16.71	
AGH1	15.59	15.45	14.66	16.82	16.56	16.65	
AGH2	15.46	15.29	15.15	16.09	16.74	16.43	
KSH	25.79	29.01	30.84	25.70	28.95	30.17	
NN	L1	31.48	35.41	36.79	31.48	37.24	38.03
	L2	45.42	49.88	50.46	49.20	50.16	53.01
Raw	19.16			19.19			

Table 8.3 (bottom) shows the performance on cross-modal retrieval. Figure 8.3 shows examples of query and database descriptors in this setting and their corresponding binary codes. NN-based methods significantly outperform CM-SSH. Furthermore, MM-NN shows superior performance compared to CM-NN, which can be explained by the use of intra-modal training data in addition to inter-modal one. Applying the hash functions trained in the multimodal set-

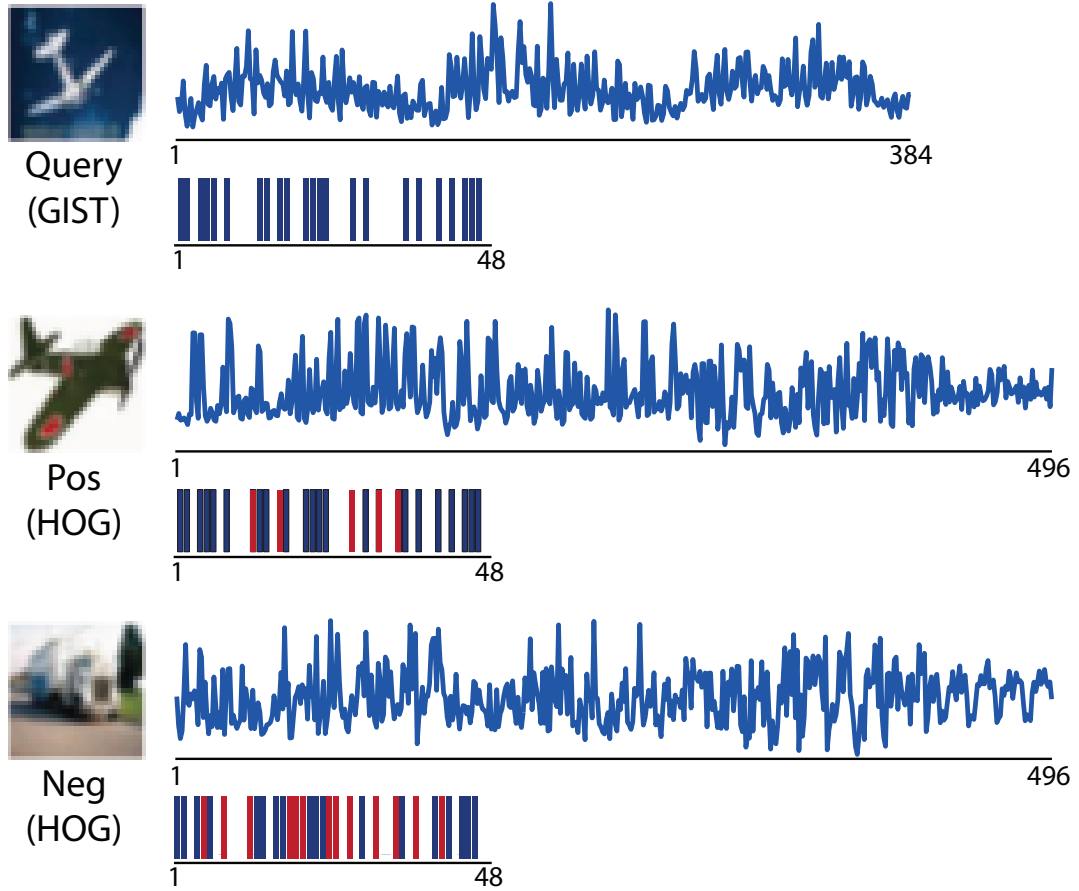


Figure 8.3. Example of GIST-HOG matching on CIFAR dataset. Shown are the original descriptors and their 48-bit MM-NN L2 hash codes. Red shows the bits that are different w.r.t. the query.

ting to unimodal retrieval (GIST-GIST and HOG-HOG in Table 8.3), MM hash achieves slightly better performance compared to the corresponding results obtained with unimodal training shown in Table 8.2. This result supports the hypothesis of multimodal information as a kind of regularization in training. Figure 8.4 (left) shows the precision recall curves for the cross-modal retrieval cases.

Table 8.3. Unimodal and cross-modal retrieval experiment on the CIFAR10 dataset. All methods were trained using multimodal data. CCA produces Euclidean embeddings. Performance is shown as mAP in %.

GIST – GIST					HOG – HOG		
Method / $m$	12	24	48		12	24	48
CCA	11.21	11.73	12.36		10.26	10.44	10.83
CM-SSH	16.93	16.78	16.17		17.65	17.60	17.50
CM-NN	L1	24.71	28.82	31.34	25.10	29.23	32.55
	L2	41.60	45.23	44.22	47.15	45.11	44.25
MM-NN	L1	28.49	34.31	34.33	30.64	36.11	36.01
	L2	<b>46.62</b>	<b>48.62</b>	<b>52.00</b>	<b>49.46</b>	<b>52.34</b>	<b>53.40</b>

GIST – HOG					HOG – GIST		
Method / $m$	12	24	48		12	24	48
CCA	10.04	10.06	10.09		10.21	10.40	10.84
CM-SSH	17.21	15.83	14.44		17.28	17.04	16.62
CM-NN	L1	24.56	28.38	32.72	25.11	29.30	33.25
	L1	47.89	47.52	47.09	43.05	45.79	45.32
MM-NN	L1	29.53	35.00	35.39	29.11	35.26	35.07
	L2	<b>48.97</b>	<b>51.15</b>	<b>54.01</b>	<b>46.80</b>	<b>49.97</b>	<b>51.06</b>



### 8.3.2 NUS

NUS [Chua et al., 2009] is a multi-class dataset containing annotated images from Flickr<sup>2</sup>. The images have been manually categorized into 81 classes (one image can belong to more than a single class) and are represented as 500-dimensional bags of SIFT features (BoF, used as the first modality) and 1000-dimensional bags of text tags (Tags, used as the second modality). To produce results consistent with the state-of-the-art, we follow the dataset generation protocol of Liu et al. [2011], which considers only the top-21 most frequent classes and used 5K samples to train KSH. Full mAP and mAP@10 was used as the retrieval quality criteria.

Table 8.4 shows the unimodal performance of several hashing methods of different lengths, where MM-NN hash outperforms the best competitor. Due to the ambiguous nature of this multi-class dataset, we did not experience improvements using an additional layer. Notice also that KSH performs worse than AGH, a completely unsupervised technique. This can be attributed to the inability of binary labels to discriminate the various degrees of similarity given by class intersection; we believe that trivial and less generalizable solutions are favored with such setup. We intend to further investigate the multi-class problem in future work.

Table 8.5-(bottom) reports the performance of the several methods using hashes up to 256bit. CCA is used as Euclidean baseline also in this case. The NN-based methods outperform CM-SSH by large margin while still keeping almost the same code generation complexity. Figure 8.4-(right) shows the precision-recall curve for the cross modal retrieval, MM-NN delivers the best performance.

Figure 8.5 shows cross-modal retrieval results using artificially created Tags vectors containing specific words as queries. These Tags are hashed using  $\eta$  and matched to BoFs hashed using  $\xi$ . The retrieved results are meaningful and most of them belong to the same class. The results produced by NN hash (bottom) are visually more meaningful compared to CM-SSH (top). Figure 8.6 shows image annotation results, the top five Tags matches from a BoF query are retrieved and assigned the corresponding image annotations.

---

<sup>2</sup><https://www.flickr.com>

Table 8.4. Unimodal training and retrieval experiment on the NUS dataset. NN hash was trained on single modality only. Performance is shown as mAP@10 / mAP in %, (– indicates no convergence was reached).

Method / $m$	BoF – BoF		
	16	64	256
<b>DiffHash</b>	54.70 / 37.40	52.97 / 36.88	53.71 / 36.75
<b>SSH</b>	45.31 / 43.76	59.00 / 43.40	59.58 / 42.21
<b>AGH1</b>	54.53 / 38.31	59.38 / 38.09	– / –
<b>AGH2</b>	53.86 / 38.24	59.56 / 39.08	– / –
<b>KSH</b>	56.25 / 49.84	64.25 / 51.30	66.46 / 51.78
<b>NN</b>	<b>60.93 / 53.40</b>	<b>66.52 / 57.10</b>	<b>72.57 / 59.36</b>
<b>Raw</b>	61.53 / 38.73		

Method / $m$	Tags – Tags		
	16	64	256
<b>DiffHash</b>	72.85 / 42.45	80.97 / 41.02	79.82 / 39.58
<b>SSH</b>	44.51 / 41.99	61.54 / 44.23	70.26 / 45.09
<b>AGH1</b>	74.60 / 45.37	79.07 / 41.52	– / –
<b>AGH2</b>	67.60 / 47.55	77.99 / 43.29	– / –
<b>KSH</b>	72.25 / 60.11	70.29 / 57.69	84.05 / 62.68
<b>NN</b>	<b>79.25 / 65.96</b>	<b>83.87 / 68.04</b>	<b>87.08 / 67.40</b>
<b>Raw</b>	83.02 / 35.20		

Table 8.5. Unimodal and cross-modal retrieval experiment on the NUS dataset. All methods were trained using multimodal data. CCA produces Euclidean embeddings. Performance is shown as mAP@10 / mAP in %.

BoF – BoF				
Method / $m$	16	64	256	
CCA	58.72 / 42.26	61.58 / 43.26	63.51 / 43.94	
CM-SSH	41.23 / 44.69	50.30 / 43.45	53.23 / 41.56	
CM-NN	52.16 / 50.34	64.33 / 51.44	67.55 / 50.14	
MM-NN	<b>60.02 / 53.09</b>	<b>64.66 / 51.87</b>	<b>70.45 / 57.84</b>	

Tags – Tags				
Method / $m$	16	64	256	
CCA	77.66 / 42.50	81.79 / 38.71	81.64 / 37.87	
CM-SSH	71.33 / 48.74	80.11 / 49.80	83.00 / 47.62	
CM-NN	75.18 / 61.70	79.62 / 61.21	83.44 / 64.68	
MM-NN	<b>78.99 / 65.52</b>	<b>83.31 / 64.64</b>	<b>86.79 / 69.40</b>	

BoF – Tags				
Method / $m$	16	64	256	
CCA	35.75 / 34.35	39.17 / 35.84	32.00 / 36.79	
CM-SSH	61.63 / 47.78	62.08 / 44.61	61.18 / 40.61	
CM-NN	<b>70.07 / 57.16</b>	<b>72.86 / 58.44</b>	<b>74.83 / 60.28</b>	
MM-NN	<b>64.57 / 57.44</b>	68.07 / 56.33	<b>73.12 / 61.63</b>	

Tags – BoF				
Method / $m$	16	64	256	
CCA	35.72 / 35.16	48.33 / 40.46	61.52 / 43.11	
CM-SSH	55.48 / 45.98	59.10 / 46.87	55.83 / 45.31	
CM-NN	70.10 / <b>57.17</b>	71.56 / <b>57.70</b>	76.74 / 59.59	
MM-NN	<b>73.40 / 56.91</b>	<b>73.28 / 55.83</b>	<b>78.77 / 61.09</b>	

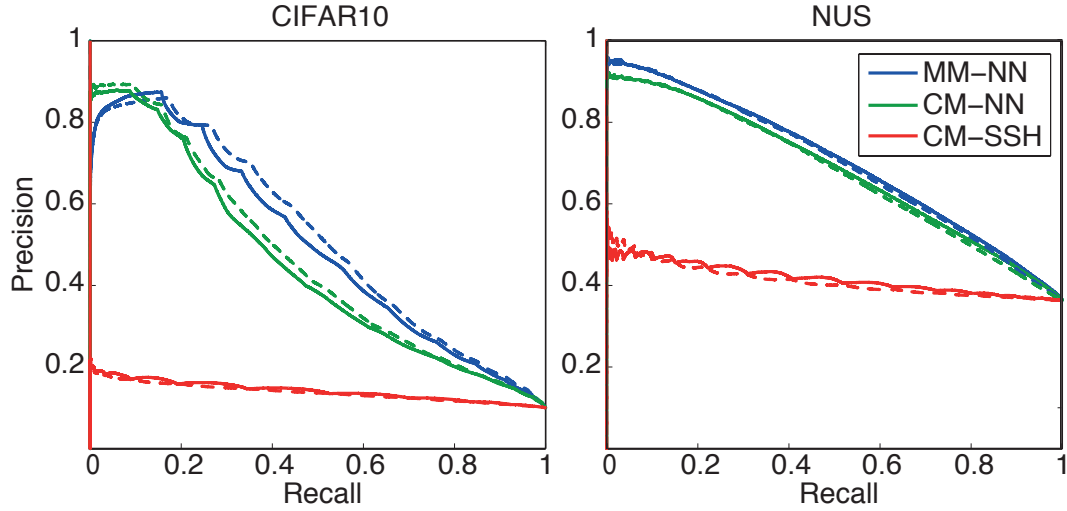


Figure 8.4. Precision-Recall curves for the cross-modal retrieval experiments on CIFAR10 (solid: HOG–GIST, dashed: GIST–HOG) and NUS (solid: Tag–Bof, dashed: Bof–Tag).

### 8.3.3 Wiki

In the third experiment, the results of [Rasiwasia et al. \[2010\]](#), using the dataset of 2866 annotated images from Wikipedia, were reproduced. The images are categorized in 10 classes and represented as 128-dimensional bags of SIFT features (Image modality) and 10-dimensional LDA topic model (Text modality). Table 8.6 shows the mAP for the Image–Text and Text–Image cross-modal retrieval experiment. For reference, we also reproduce the results reported in [Rasiwasia et al. \[2010\]](#) using correlation matching (CM), semantic matching (SM), and semantic correlation matching (SCM). MM-NN largely outperforms SCM in all evaluation criterion with codes that are at least  $10\times$  smaller and that can be searched very efficiently. We should stress however that these results are not directly comparable with ours: while [\[Rasiwasia et al., 2010\]](#) find a Euclidean embedding, we use Hamming embedding (in general, a more difficult problem). Figure 8.8 shows a few matching examples.



Figure 8.5. Example of text-based image retrieval on NUS dataset using multi-modal hashing. Shown are top five image matches produced by CM-SSH (odd rows) and MM-NN (even rows) in response to three different textual queries.

Table 8.6. Cross-modal retrieval experiment on the Wiki dataset using 32-bit hashes (L2 with 32  $\tanh$  units) and Euclidean embeddings from [Rasiwasia et al. \[2010\]](#) (marked with \*).

		Image-Text	Text-Image	Avg
	<b>CM-SSH</b>	22.2	18.4	20.3
	<b>CM*</b>	24.9	19.6	22.3
	<b>SM*</b>	22.5	22.3	22.4
	<b>SCM*</b>	27.7	22.6	25.2
<b>MM-NN</b>	<b>L1</b>	37.8	24.7	31.2
	<b>L2</b>	<b>57.5</b>	<b>27.4</b>	<b>42.4</b>
<b>CM-NN</b>	<b>L1</b>	32.6	23.2	25.5
	<b>L2</b>	48.5	25.8	37.1

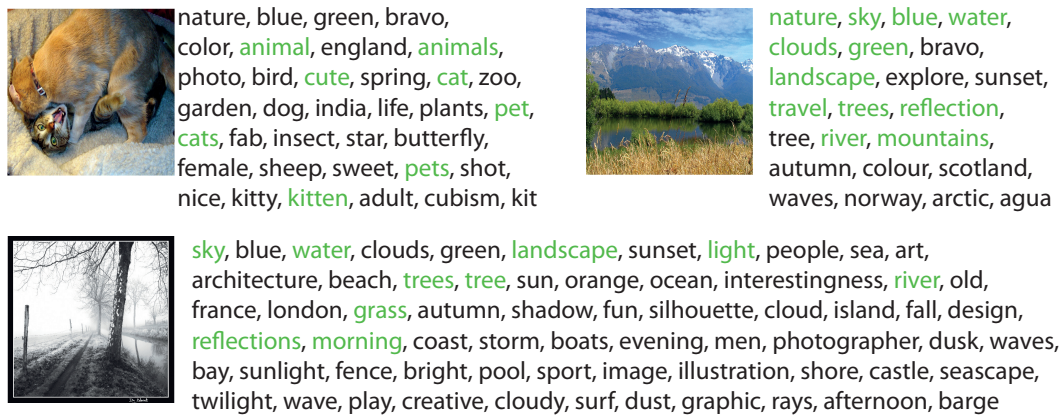


Figure 8.6. Example of image annotation on the NUS dataset using multimodal hashing. Shown are Tags returned for the image query on the left. Groundtruth tags are shown in green.

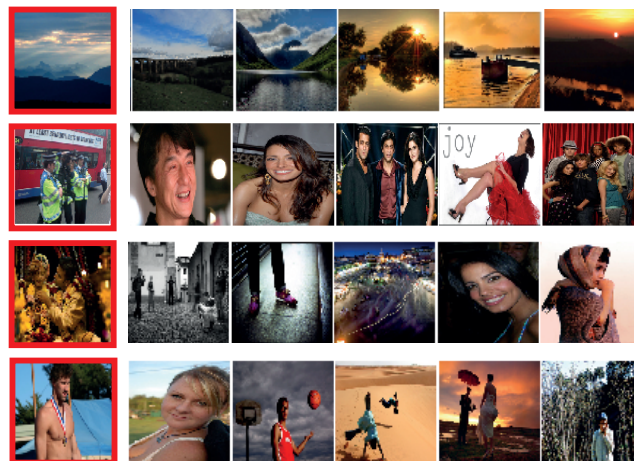


Figure 8.7. Cross-modal (Bof-Tags) retrieval on the NUS dataset. Shown are top five matches different image queries (marked in red), ranked according to Tags similarity using 64-bit MM-NN hash.



Figure 8.8. Cross-modal (Image–Text) retrieval on the Wiki dataset. Shown are top five matches different image queries (marked in red), ranked according to text similarity using 32-bit MM-NN hash.





## Chapter 9

# Conclusions and Future Research

This PhD Thesis has presented advances in deep learning for vision applications targeting the automatic surface inspection system of steel industry. Some of the presented works will be evaluated and tested at ArcelorMittal which has already started internal trials.

In summary we have developed algorithms and methodologies which achieve state-of-the-art performance for generic steel defect classification and segmentation and that are largely applicable to other problems as well.

Mathematical Morphology has been finally introduced to the deep learning community thought a simple yet powerful formulation which is flexible and advantageous in comparison with other approaches. It does not require to a priori fix the operator to be learned or the structuring element and can learn them both through gradient descent.

The dissertation ends with a novel multimodal similarity sensitive hashing method which is easily generalizable to many modalities, not just two as in other published approaches, and attains outstanding performance compared to previous state-of-the-art.

We believe that further improvements can be achieved and we foresee several future developments for our investigation, here briefly listed.

### 9.1 Learning compact key-point image descriptors

In the decade since the seminal work of Lowe [2004], SIFT-like feature descriptors [Mikolajczyk et al., 2005; Bay et al., 2008; Tola et al., 2010] have become ubiquitous in the computer vision and pattern recognition literature. One of the main reasons for the success of SIFT is that it is “good enough” in many image analysis applications, such as stereo matching.

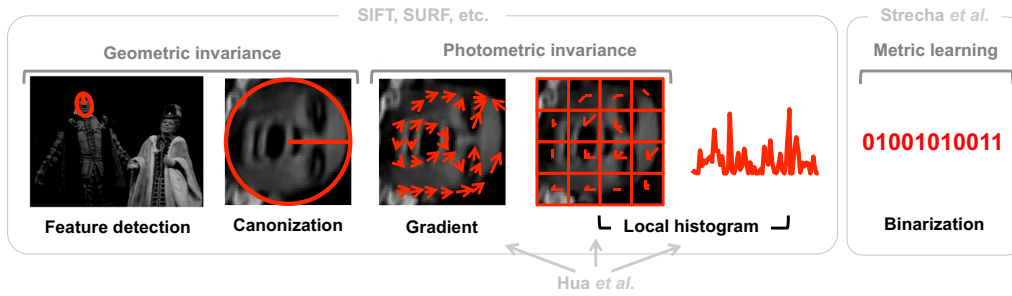


Figure 9.1. Typical pipeline of feature descriptor construction, consisting of: affine-invariant region detection and canonization, ensuring approximate invariance to view point transformations; linear filtering part (e.g. gradient computation in SIFT), ensuring illumination invariance; non-linear part (e.g. local directions histogram in SIFT). Hua et al. focused on tuning the parameters of the linear and non-linear parts of SIFT. Strecha et al. added another binarization stage.

A schematic pipeline of SIFT-like feature descriptors is depicted in Figure 9.1. Typically, the first stage consists of affine-invariant region detection. Locally, viewpoint transformations can be approximated as affine transformations, hence affine invariance ensures approximate viewpoint invariance [Mikolajczyk and Schmid, 2002]. Secondly, the detected regions undergo “canonization” that undoes the effect of the transformation and normalizes the content of the region into a patch of fixed size (typically  $32 \times 32$  or  $64 \times 64$  pixels), also fixing the orientation. The third (linear) stage in SIFT is the computation of the gradients in the patch. Gradients capture the photometric discontinuities (edges) in the image and are known to be insensitive to illumination variations. Finally, at the fourth (non-linear) stage, the gradient directions are quantized and aggregated into a localized directions histogram. The classical configuration of SIFT used 8 directions and a histogram in  $4 \times 4$  bins, producing a 128-dimensional descriptor vector.

Each of the stages in the aforementioned pipeline might have different variants and implementations [Mikolajczyk and Schmid, 2004; Bay et al., 2008]. However, most of these choices are rather heuristics and there is no clear reason why, e.g., one way of filtering or histogram computation should be preferred over another. One of the attempts to question that certain stages of the pipeline are arbitrary was done by Hua et al. [2007] and other authors [Winder and Brown, 2007; Winder et al., 2009; Chandrasekhar et al., 2009; Brown et al., 2010].

Several attempts have been done to “post-process” SIFT-like descriptors (adding a fifth stage in the pipeline in Figure 9.1), for example to achieve lower-dimensional feature vectors without significantly compromising the performance, using dimensionality reduction techniques [Mikolajczyk and Schmid, 2003; Mikolajczyk and Matas, 2007; Winder et al., 2009; Brown et al., 2010], or quantization [Tuytelaars and Schmid, 2007; Chandrasekhar et al., 2009; Brown et al., 2010]. Strecha et al. [2012] proposed adding a fifth *binarization* stage of the descriptor vector. They showed that such a binarization can be performed using similarity-preserving hashing [Gionis et al., 1999; Shakhnarovich, 2005; Weiss et al., 2008; Kulis and Darrell, 2009; Raginsky and Lazebnik, 2009].

In light of these results, it is appropriate to ask a rather philosophical question whether invariant feature descriptors should be constructed or rather learned. The traditional computer vision approaches have sided with “invariance by construction”, the best manifestation of which can be seen in the first two stages (feature detection) of the pipeline in Figure 9.1, where affine-invariant regions have proven theoretical properties. However, in real situations such invariance is only approximate – first, as already mentioned, because affine transformations are only a local approximations of real perspective transformations, and second, since real optical systems have imperfections, and geometric transformations do not commute with blur [Kimmel et al., 2011]. Other types of transformations such as lens distortion or motion blur are more difficult or even impossible to model. Thus, the best that can be achieved by the “invariance by construction” paradigm is only approximate invariance. An opposite approach is to learn task-related features from scratch, such as done in the works by LeCun [Ranzato et al., 2007c] and other authors [Taylor et al., 2011], which can be labeled as “invariance by learning”.

We propose to take the average of the two approaches, arguing that feature detection is good enough in many cases to take advantage of (approximate) viewpoint invariance. We map stages 3–5 of the pipeline to a siamese MPCNN similarly to what we proposed in Masci et al. [2011b] which allows the construction of compact binary descriptors directly from patches of raw pixels. Our approach can hence be considered a generalization of Strecha et al. [2012], and their approach is obtained as a particular setting of our pipeline without the convolutional and pooling layers.

We are currently thoroughly evaluating our proposed framework on a large set of benchmarks. Preliminary results confirm our belief that learning directly from raw pixel intensities is highly beneficial.

## 9.2 Morphological Operators for Structured Pooling

The pooling stage of a CNN is crucial to obtain good performance for segmentation and classification tasks. Therefore there have been extensive studies on the kind of operation and on its several variants [Zeiler and Fergus, 2013a; Scherer et al., 2010; Sermanet et al., 2012]. In particular it seems that sometimes a max-pooling operation, even though remarkably efficient, can be slightly outperformed in small models by smoothed versions of it. Sermanet et al. [2012] have extensively evaluated this approach. However, the selection of the best parameter for the  $\ell_p$  pooling involves a trial and error process which in case of large models and datasets takes lot of time.

With our recent development in learning morphological operators [Masci et al., 2013a] we find striking the similarity between the max-pooling operation and the morphological dilate with a fixed structuring element. Indeed max-pooling could be replaced by a dilate followed by a downsampling operator which takes a pixel for every non overlapping portion of the image. One of the major consequences of our contribution is the possibility to learn, with the counter-harmonic mean formulation, the dilate with all its smoothed variants in conjunction with the structuring element. In fact, even if in the formulation of the pooling operation of Sermanet et al. [2012] the learning of the parameter  $P$  would be introduced, there will be still no possibility to learn its morphological structure.

Using our morphological layer will make possible to learn a pooling operation which takes into account not only the strength of the features but also their structural properties, therefore introducing a much more flexible way of selecting features.

In the remaining of our studies we intend to investigate such addition in tasks such as classification and segmentation.

# Bibliography

- AccelerEyes. ArrayFire v1.2. <http://accelereyes.com>, 2012.
- Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. *Ecole Polytechnique Fédéral de Lausanne (EPFL), Tech. Rep*, 2:3, 2010.
- Timo Ahonen, Jiří Matas, Chu He, and Matti Pietikäinen. Rotation invariant image description with local binary pattern histogram fourier features. In *Proceedings of the 16th Scandinavian Conference on Image Analysis, SCIA '09*, pages 61–70, Berlin, Heidelberg, 2009. Springer-Verlag.
- Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.
- Jesús Angulo. *Morphologie mathématique et indexation d'images couler. Application á la microscopie en biomédecine*. PhD thesis, École des Mines de Paris, 2003.
- Jesús Angulo. Segmentation morphologique d'images multivariées: de la couleur aux images hyperspectrales. In *Lecture Notes for Ecole d'Hiver pour l'Image Num'érique Couleur*, 2009.
- Jesús Angulo. Pseudo-morphological image diffusion using the counter-harmonic paradigm. In *Proc. of ACTVS'2010, LNCS Vol. 6474, Part I*, pages 426–437. Springer, 2010.
- Timur Ash. Dynamic node creation in backpropagation neural networks. *Connection Science*, 1(4):365–375, 1989.
- Olugbenga Ayinde and Yee-Hong Yang. Face recognition approach based on rank correlation of gabor-filtered images. *Pattern Recognition*, 35(35):1275–1289, 2002.

- Francis R. Bach, Gert RG Lanckriet, and Michael I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.
- Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.
- Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.
- Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
- Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.
- Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems (NIPS)*, 2007.
- Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.
- Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *CoRR*, abs/1305.6663, 2013.
- James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.
- Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- Adriana Bodnarova, Mohammed Bennamoun, and Shane Latham. Optimal gabor filters for textile flaw detection. *Pattern recognition*, 35(12):2973–2991, 2002.

- Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005.
- Anna Bosch, Andrew Zisserman, and Xavier Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 401–408. ACM, 2007.
- Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.
- Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.
- J. Bromley, I. Guyon, Y. Lecun, E. Säckinger, and R. Shah. Signature verification using a "siamese" time delay neural network. In *Proc. NIPS*, 1994.
- Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4), August 1993.
- Alexander M. Bronstein, Pablo Sprechmann, and Guillermo Sapiro. Learning efficient structured sparse models. In *ICML*, 2012.
- Michael M. Bronstein, Alexander M. Bronstein, Fabrice Michel, and Nikos Paragios. Data fusion through cross-modality metric learning using similarity-sensitive hashing. In *Proc. CVPR*, 2010.
- Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 99(PrePrints), 2010.
- Arthur E. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proceedings of the Harvard University Symposium on Digital Computers and Their Applications*, 1961.
- Peter S. Bullen. *Handbook of Means and Their Inequalities, 2nd Edition*. Springer, 1987.
- Albert Cardona, Stephan Saalfeld, Stephan Preibisch, Benjamin Schmid, Anchi Cheng, Jim Pulkas, Pavel Tomancak, and Volker Hartenstein. An integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLoS Biol*, 8, 10 2010.

- Vijay Chandrasekhar, Gabriel Takacs, David M. Chen, Sam S. Tsai, Radek Grzeszczuk, and Bernd Girod. Chog: Compressed histogram of gradients a low bit-rate feature descriptor. In *CVPR*, pages 2504–2511, 2009.
- Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- Ken Chatfield, Victor Lempitsky, Andrea Vedaldi, and Andrew Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *British Machine Vision Conference*, 2011.
- Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *J. Mach. Learn. Res.*, 11:1109–1135, March 2010. ISSN 1532-4435.
- Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, page 48. ACM, 2009.
- Dan C. Ciresan, Ueli Meier, Luca M. Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.
- Dan C. Ciresan, Ueli Meier, Luca M. Gambardella, and Jürgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *ICDAR*, pages 1250–1254, 2011.
- Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks (IJCNN2011)*, pages 1918–1921, 2011a.
- Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *International Joint Conference on Artificial Intelligence (IJCAI2011)*, pages 1237–1242, 2011b.
- Dan C. Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, pages 2852–2860, 2012a.



- Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012b.
- Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition*, pages 3642–3649, 2012c.
- Dan C. Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images using deep neural networks. In *MICCAI 2013*, 2013a.
- Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *MICCAI*, volume 2, pages 411–418, 2013b.
- Adam Coates, Honglak Lee, and Andrew Ng. An analysis of single-layer networks in unsupervised feature learning. In *Advances in Neural Information Processing Systems*, 2010.
- Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.
- Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
- José Crespo, Ronald W Schafer, Jean Serra, Cristophe Gratin, and Fernand Meyer. The flat zone approach: a general low-level region merging segmentation method. *Signal Processing*, 62(1):37–60, 1997.
- Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.
- Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.
- Kristin J Dana, Bram Van Ginneken, Shree K Nayar, and Jan J Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics (TOG)*, 18(1):1–34, 1999.

- Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.
- Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
- Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962.
- Michael Elad. *Sparse and redundant representations*. Springer, 2010.
- Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, and Pascal Vincent. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, 11:625–660, 2010.
- Scott E. Fahlman. The recurrent cascade-correlation learning algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, pages 190–196, 1991.
- Clément Farabet, Cyril Poulet, and Yann LeCun. An fpga-based stream processor for embedded real-time vision with convolutional networks. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 878–885. IEEE, 2009.
- Clément Farabet, Yann LeCun, Koray Kavukcuoglu, Eugenio Culurciello, Berin Martini, Polina Akselrod, and Selcuk Talay. *Large-scale FPGA-based convolutional networks*. Cambridge, UK: Cambridge University Press, 2011.
- Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013. in press.
- Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *IEEE CVPR Workshop of Generative Model Based Vision (WGMBV)*, 2004.
- Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

- Kunihiko Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position - neocognitron. In *Trans. IECE*, 1979.
- Kunihiko Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.
- Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proc. Conf. Very Large Data Bases*, pages 518–529, 1999.
- Ross Girshick, Jeff Donahue, Trevor Darrell, and Malik Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.
- Alessandro Giusti, Dan C. Ciresan, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. In *International Conference on Image Processing (ICIP13)*, 2013.
- Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.
- Tom Goldstein and Stanley Osher. The split bregman method for l1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.
- Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 817–824. IEEE, 2011.
- Yunchao Gong, Sanjiv Kumar, Vishal Verma, and Svetlana Lazebnik. Angular quantization-based binary codes for fast similarity search. In *Proc. NIPS*, pages 1205–1213, 2012.
- Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013a.
- Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *ICML*, 2013b.

- Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD thesis, Technische Universität München, 2008.
- Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural nets. In *ICML06: Proceedings of the International Conference on Machine Learning*, 2006.
- Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.
- Emmanuèle Grosicki and Haikal El-Abed. ICDAR 2011 - french handwriting recognition competition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1459–1463. IEEE, 2011.
- Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press, 2006.
- S. J. Hanson and L. Y. Pratt. Some comparisons of constraints for minimal network construction with back-propagation. In *NIPS*, volume 1, 1988.
- Neal R. Harvey and Stephen Marshall. The use of genetic algorithms in morphological filter design. *Signal Processing: Image Communication*, 8(1):55–71, 1996.
- Henk JAM Heijmans. Mathematical morphology: a modern approach in image processing based on algebra and geometry. *SIAM review*, 37(1):1–36, 1995.
- Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.
- Sepp Hochreiter and Jürgen Schmidhuber. Feature extraction through LO-COCODE. *Neural Computation*, 11(3):679–714, 1999.

- Gang Hua, Matthew Brown, and Simon Winder. Discriminant embedding for local image descriptors. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.
- David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, March 1968.
- Viren Jain, Joseph F Murray, Fabian Roth, Srinivas Turaga, Valentin Zhigulin, Kevin L Briggman, Moritz N Helmstaedter, Winfried Denk, and H Sebastian Seung. Supervised learning of image restoration with convolutional networks. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.
- Kevin Jarrett, Koray Kavukcuoglu, Marc’Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV’09)*. IEEE, 2009.
- Richard Arnold Johnson and Dean W. Wichern. *Applied multivariate statistical analysis*, volume 5. Prentice hall Upper Saddle River, NJ, 2002.
- Koray Kavukcuoglu. *Learning Feature Hierarchies for Object Recognition*. PhD thesis, New York University, 2011.
- Koray Kavukcuoglu, Marc’Aurelio Ranzato, and Yann Lecun. Fast inference in sparse coding algorithms with applications to object recognition. Technical Report CBLL-TR-2008-12-01, Computational and Biological Learning Lab, Courant Institute, NYU, 2008.
- Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, and Yann LeCun. Learning convolutional feature hierachies for visual recognition. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
- Henry J. Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30(10): 947–954, 1960.
- Ron Kimmel, Cuiping Zhang, Alexander M. Bronstein, and Michael M. Bronstein. Are msr features really interesting? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(11):2316–2320, 2011.

- Simon Korman and Shai Avidan. Coherency sensitive hashing. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1607–1614. IEEE, 2011.
- Shawn Kraut, Louis L. Scharf, and Ronald W. Butler. The adaptive coherence estimator: a uniformly most-powerful-invariant adaptive detection statistic. *Signal Processing, IEEE Transactions on*, 53(2):427–438, 2005.
- Alex Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, Computer Science Department, University of Toronto, 2009.
- Alex Krizhevsky. Convolutional deep belief networks on CIFAR-10. *Unpublished manuscript*, 2010.
- Alex Krizhevsky. cuda-convnet. <https://code.google.com/p/cuda-convnet/>, 2011.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 1, page 4, 2012.
- Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.
- Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2130–2137. IEEE, 2009.
- Ajay Kumar and Grantham K.H. Pang. Defect detection in textured materials using gabor filters. *IEEE Transactions on Industry Application*, 38(2):425–440, 2002.
- Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.
- Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision, 2006 IEEE 9th International Conference on*, pages 2169–2178, 2006.

- Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *ICML*, 2012.
- Quoc V Le, Jiquan Ngiam, Zhenghao Chen, Daniel Jin hao Chia, Pang Wei Koh, Andrew Y Ng, and D Chia. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1279–1287, 2010.
- Yann LeCun. *Modeles connexionnistes de l'apprentissage (connectionist learning models)*. PhD thesis, Université P et M. Curie (Paris 6), June 1987.
- Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, R.E. Howard, Wayne Hubbard, and D. Lawrence Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989a.
- Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPS*, volume 2, pages 598–605, 1989b.
- Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, R.E. Howard, Wayne Hubbard, and D. Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS\*89)*, Denver, CO, 1990. Morgan Kaufman.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, November 1998.
- Daewon Lee, Matthias Hofmann, Florian Steinke, Yasemin Altun, Nathan D Cahill, and Bernhard Scholkopf. Learning similarity measure for multi-modal 3d image registration. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 186–193. IEEE, 2009a.
- Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.
- Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area V2. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

- Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009b.
- Thomas K. Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.
- Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master’s thesis, Univ. Helsinki, 1970.
- Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1–8, 2011.
- Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012.
- David .G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.
- David G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- Dimitris Manolakis, David Marden, and Gary A Shaw. Hyperspectral image processing for automatic target detection applications. *Lincoln Laboratory Journal*, 14(1):79–116, 2003.
- Luiz AO Martins, Flávio LC Pádua, and Paulo EM Almeida. Automatic detection of surface defects on rolled steel using computer vision and artificial neural networks. In *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, pages 1081–1086. IEEE, 2010.
- Jonathan Masci, Ueli Meier, Dan Ciresan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *ICANN (1)*, pages 52–59, 2011a.



Jonathan Masci, Davide Migliore, Michael M. Bronstein, and Jürgen Schmidhuber. Descriptor learning for omnidirectional image matching. *CoRR*, abs/1112.6291, 2011b.

Jonathan Masci, Ueli Meier, Dan Cireşan, Fricout Gabriel, and Jürgen Schmidhuber. Steel defect classification with max-pooling convolutional neural networks. In *International Joint Conference on Neural Networks (IJCNN2012)*, 2012.

Jonathan Masci, Jesús Angulo, and Jürgen Schmidhuber. A learning framework for morphological operators using counter-harmonic mean. In *Mathematical Morphology and Its Applications to Signal and Image Processing*, volume 7883 of *Lecture Notes in Computer Science*, pages 329–340. Springer Berlin Heidelberg, 2013a.

Jonathan Masci, Alessandro Giusti, Dan C. Ciresan, Gabriel Fricout, and Jürgen Schmidhuber. A fast learning algorithm for image segmentation with max-pooling convolutional networks. In *International Conference on Image Processing (ICIP13)*, 2013b.

Jonathan Masci, Ueli Meier, Fricout Gabriel, and Jürgen Schmidhuber. Multi-scale pyramidal pooling network for generic steel defect classification. In *International Joint Conference on Neural Networks (IJCNN2013)*, 2013c.

Jonathan Masci, Michael M. Bronstein, Alexander A. Bronstein, and Jürgen Schmidhuber. Multimodal similarity-preserving hashing (in press). *PAMI*, 2014a.

Jonathan Masci, Davide Migliore, Michael M. Bronstein, and Jürgen Schmidhuber. Descriptor learning for omnidirectional image matching. In Roberto Cipolla, Sebastiano Battiato, and Giovanni Maria Farinella, editors, *Registration and Recognition in Images and Videos*, volume 532 of *Studies in Computational Intelligence*, pages 49–62. Springer Berlin Heidelberg, 2014b. ISBN 978-3-642-44906-2.

Brian McFee and Gert Lanckriet. Partial order embedding with multiple kernels. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 721–728. ACM, 2009.

Brian McFee and Gert Lanckriet. Learning multi-modal similarity. *The Journal of Machine Learning Research*, 12:491–523, 2011.

- Ueli Meier, Dan C. Ciresan, Luca M. Gambardella, and Jürgen Schmidhuber. Better digit recognition with a committee of simple neural nets. In *International Conference on Document Analysis and Recognition*, pages 1135–1139, 2011.
- Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Schoelkopf, and Klaus R. Mueller. Fisher discriminant analysis with kernels. In *Proc. Neural Networks for Signal Processing*, 1999.
- Krystian Mikolajczyk and Jiri Matas. Improving descriptors for fast tree matching by optimal linear projection. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.
- Krystian Mikolajczyk and Cordelia Schmid. An affine invariant interest point detector. In *Computer Vision-ECCV 2002*, pages 128–142. Springer, 2002.
- Krystian Mikolajczyk and Cordelia Schmid. A Performance Evaluation of Local Descriptors. In *Proc. of Computer Vision and Pattern Recognition Conference (CVPR 2010)*, pages 257–263, June 2003.
- Krystian Mikolajczyk and Cordelia Schmid. A Performance Evaluation of Local Descriptors. "*IEEE Transactions on Pattern Analysis and Machine Intelligence*", 27(10):1615–1630, 2004.
- Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Timor Kadir, and Luc Van Gool. A comparison of affine region detectors. *International journal of computer vision*, 65 (1-2):43–72, 2005.
- John. E. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 847–854. Morgan Kaufmann, 1992.
- M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. Morgan Kaufmann, 1989.
- Laurent Najman and Hugues Talbot. *Mathematical Morphology*. John Wiley & Sons, 2013.

- Makoto Nakashizuka, Shinji Takenaka, and Youji Iiguni. Learning of structuring elements for morphological image model with a sparsity prior. In *IEEE International Conference on Image Processing, ICIP 2010*, pages 85–88, 2010.
- José MP Nascimento and José MB Dias. Vertex component analysis: A fast algorithm to unmix hyperspectral data. *Geoscience and Remote Sensing, IEEE Transactions on*, 43(4):898–910, 2005.
- Fabian Nasse, Christian Thureau, and Gernot A Fink. Face detection using gpu-based convolutional neural networks. In *Computer Analysis of Images and Patterns*, pages 83–90. Springer, 2009.
- Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.
- Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Leon Bottou, and Paolo Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371, September 2005. Special issue on Molecular and Cellular Bioimaging.
- Mohammad Norouzi and David M Blei. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 353–360, 2011.
- Mohammad Norouzi, David J Fleet, and Ruslan Salakhutdinov. Hamming distance metric learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.
- Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.
- Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971 –987, July 2002.
- B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, June 1996.

- Lúcio F. C. Pessoa and Petros Maragos. Mrl-filters: a general class of nonlinear systems and their optimal design for image processing. *IEEE Transactions on Image Processing*, 7(7):966–978, 1998.
- Nicolas Pinto, David Doukhan, James J. DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11):e1000579, 2009.
- G-J Qi, Charu Aggarwal, Yong Rui, Qi Tian, Shiyu Chang, and Thomas Huang. Towards cross-category knowledge propagation for learning visual concepts. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 897–904. IEEE, 2011.
- Maxim Raginsky and Svetlana Lazebnik. Locality-Sensitive Binary Codes from Shift-Invariant Kernels. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.
- Marc’Aurelio Ranzato. *Unsupervised Learning of Feature Hierarchies*. PhD thesis, New York University, 2009.
- Marc’Aurelio Ranzato and Yann LeCun. A sparse and locally shift invariant feature extractor applied to document images. In *Proc. International Conference on Document Analysis and Recognition (ICDAR)*, 2007.
- Marc’Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS 2006)*, 2006a.
- Marc’Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS’06)*, 2006b.
- Marc’Aurelio Ranzato, Y-Lan Boureau, Sumit Chopra, and Yann LeCun. A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)*, 2007a.
- Marc’Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS 2007)*, 2007b.

- Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007c.
- Nikhil Rasiwasia, Jose Costa Pereira, Emanuele Coviello, Gabriel Doyle, Gert RG Lanckriet, Roger Levy, and Nuno Vasconcelos. A new approach to cross-modal multimedia retrieval. In *Proceedings of the international conference on Multimedia*, pages 251–260. ACM, 2010.
- Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nat. Neurosci.*, 2(11):1019–1025, 1999.
- Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011.
- Roberto Rigamonti, Matthew A Brown, and Vincent Lepetit. Are sparse representations really relevant for image classification? In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1545–1552. IEEE, 2011.
- Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
- Frank Rosenblatt. *Principles of Neurodynamics*. Spartan Book, 1962.
- Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
- Sam T. Roweis, Geoffrey Hinton, and Ruslan Salakhutdinov. Neighbourhood component analysis. In *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 513–520, 2004.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
- Andreas S. Weigend David E. Rurnelhart and Bernardo A. Huberrnan. Generalization by weight-elimination with application to forecasting.

- Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.
- Philippe Salembier. Adaptive rank order based filters. *Signal Processing*, 27: 1–25, 1992a.
- Philippe Salembier. Structuring element adaptation for morphological filters. *J. of Visual Communication and Image Representation*, 3(2):115–136, 1992b.
- Philippe Jean Salembier Clairon and Michael Wilkinson. Connected operators: A review of region-based morphological image processing techniques. 2010.
- Velasco Forero Santiago. *Contributions en morphologie mathématique pour l’analyse d’images multivariées*. PhD thesis, École des Mines de Paris, 2012.
- Andrew Saxe, Pang W Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1089–1096, 2011.
- Dominik Scherer, Adreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, 2010.
- Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.
- Jürgen Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.
- Jürgen Schmidhuber and Daniel Prelinger. Discovering predictable classifications. *Neural Computation*, 5(4):625–635, 1993.
- Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Artificial Neural Networks-ICANN’97*, pages 583–588. Springer, 1997.
- Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, pages 3288–3291, 2012.
- Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013a.

- Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3626–3633. IEEE, 2013b.
- Jean Serra. *Image analysis and mathematical morphology*. Academic Press, London, 1982.
- Gregory Shakhnarovich. *Learning Task-Specific Similarity*. PhD thesis, MIT, 2005.
- Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 750–757. IEEE, 2003.
- Abhishek Sharma, Abhishek Kumar, H Daume, and David W Jacobs. Generalized multiview analysis: A discriminative latent space. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2160–2167. IEEE, 2012.
- Chunhua Shen, Junae Kim, Lei Wang, and Anton Van Den Hengel. Positive semidefinite metric learning with boosting. In *Advances in Neural Information Processing Systems (NIPS)*, volume 22, pages 629–633, 2009.
- Patrice Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition*, volume 3, pages 958–962, 2003.
- Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. 2013.
- Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE, 2003.
- Richard Socher, Eric H. Huang, Jeffrey Pennin, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems 24*, 2011.
- Pierre Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag Berlin, 1999.

- Pierre Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition, 2003. ISBN 3540429883.
- Pablo Sprechmann, Roei Litman, Tal B. Yakar, Alexander M. Bronstein, and Guillermo Sapiro. Supervised sparse analysis and synthesis operators. In *Advances in Neural Information Processing Systems*, pages 908–916, 2013.
- Rupesh K. Srivastava, Jonathan Masci, Sohrab Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.
- Christoph Strecha, Alexander M Bronstein, Michael M Bronstein, and Pascal Fua. LDAHash: Improved matching with smaller descriptors. *PAMI*, 34(1):66–78, 2012.
- Graham W Taylor, Ian Spiro, Christoph Bregler, and Rob Fergus. Learning invariance through imitation. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2729–2736. IEEE, 2011.
- Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(5):815–830, 2010.
- A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *PAMI*, 30(11):1958–1970, 2008a.
- Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008b.
- Srinivas Turaga, Kevin Briggman, Moritz Helmstaedter, Winfried Denk, and Sebastian Seung. Maximin affinity learning of image segmentation. In *Advances in Neural Information Processing Systems 22*, 2009.
- Srinivas C Turaga, Joseph F Murray, Viren Jain, Fabian Roth, Moritz Helmstaedter, Kevin Briggman, Winfried Denk, and H Sebastian Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural computation*, 22(2):511–38, 2010.
- Tinne Tuytelaars and Cordelia Schmid. Vector quantizing feature space with a regular lattice. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.



- Koen EA Van De Sande, Theo Gevers, and Cees GM Snoek. Evaluating color descriptors for object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1582–1596, 2010.
- Lucas J. van Vliet. Robust local max-min filters by normalized power-weighted filtering. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 1, pages 696–699. IEEE, 2004.
- Manik Varma and Andrew Zisserman. Texture classification: Are filter banks necessary? In *Computer vision and pattern recognition, 2003. Proceedings. 2003 IEEE computer society conference on*, volume 2, pages II–691. IEEE, 2003.
- Manik Varma and Andrew Zisserman. A statistical approach to texture classification from single images. *International Journal of Computer Vision*, 62(1-2): 61–81, 2005.
- Manik Varma and Andrew Zisserman. A statistical approach to material classification using image patch exemplars. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(11):2032–2047, 2009.
- Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms. <http://www.vlfeat.org/>, 2008.
- Luc Vincent. Morphological area openings and closings for grey-scale images. In *Shape in Picture*, pages 197–208. Springer, 1994.
- Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Neural Information Processing Systems (NIPS)*, 2008.
- Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 9999:3371–3408, 2010.
- Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.
- Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010a.

- Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1127–1134, 2010b.
- Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244, 2009.
- Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems (NIPS)*, volume 9, page 6, 2008.
- Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.
- Paul J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770, 1981.
- Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1): 21–35, 2010.
- Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280, 1989.
- Stephen S. Wilson. Training structuring elements in morphological networks. In *Mathematical Morphology in Image Processing, Chapter 1*, pages 1–42. Marcel Dekker, 1993.
- Simon Winder, Gang Hua, and Matthew Brown. Picking the best daisy. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 178–185. IEEE, 2009.
- Simon AJ Winder and Matthew Brown. Learning local image descriptors. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.
- Eric P Xing, Andrew Y Ng, Michael I Jordan, and Stuart Russell. Distance metric learning with application to clustering with side-information. In *Proc. NIPS*, 2002.
- Wei Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *CoRR*, abs/1107.2490, 2011.

- Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *in IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, 2009.
- Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013a.
- Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013b.
- Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional Networks. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'2010)*, 2010.
- Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pages 2018–2025. IEEE, November 2011.
- Lin Zhang, Lei Zhang, Zhenhua Guo, and David Zhang. Monogenic-lbp: A new approach for rotation invariant texture classification. In *ICIP*, pages 2677–2680, 2010.