Deep learning for texture and dynamic texture analysis

Vincent Andrearczyk

Supervisor: Prof. Paul F. Whelan

School of Electronic Engineering
Dublin City University

This dissertation is submitted for the degree of
Doctor of Philosophy

September 2017

Declaration

Vincent Andrearczyk
September 2017

Acknowledgements

I would like to first thank Prof. Paul Whelan for giving me the opportunity to pursue this research program and for his support and guidance throughout the past four years. I also want to express my gratitude to Dr. Ovidiu Ghita for his precious advice and interest in my work during the first two years.

I wish to thank all the members of the Centre for Image Processing and Analysis (CIPA) for making this period enjoyable and memorable. In particular, I thank Dr. Tony Marrero and Dr. Ram Prasad for sharing their valuable experience and suggestions. I am also grateful to my friends and former colleagues in DCU Chris and Vidak for their help.

I would like to thank my brother and my family for the precious time spent at home and in Castres, as well as my friends Florian, Kevin, Pierre, Sebastien, Thibaud, and Xavier.

I want to offer a special thank to Sonja whose support, love, patience, and help have been essential to the completion of this work.

Enfin, je remercie ma mère pour son soutien et son amour inconditionnel, dépourvu de jugement et d'attentes. Son état d'esprit et sa bonté sont de riches sources d'inspiration.

Journal Publications and Pre-prints

Vincent Andrearczyk and Paul F. Whelan (2017), Texture segmentation with Fully Convolutional Networks", arXiv preprint arXiv:1703.05230.

Vincent Andrearczyk and Paul F. Whelan (2017), Convolutional Neural Network on Three Orthogonal Planes for Dynamic Texture Classification, Under review at *Pattern Recognition*, subject to minor revisions, arXiv preprint arXiv:1703.05530.

Vincent Andrearczyk and Paul F. Whelan (2016), Using Filter Banks in Convolutional Neural Networks for Texture Classification, *Pattern Recognition Letters*, vol. 84, pp. 63–69.

Book Chapter Publication

Vincent Andrearczyk and Paul F. Whelan (2017), Deep Learning in Texture Analysis and its Application to Tissue Image Classification, *Biomedical Texture Analysis (BTA)*, Fundamentals, Tools and Challenges (Academic Press, London, 2017), Editors: Adrien Depeursinge, Omar S. Al-Kadi and J. Ross Mitchell, In Press, Published Date: 1st October 2017.

Conference Publications

Vincent Andrearczyk and Paul F. Whelan (2016), Deep Learning for Biomedical Texture Image Analysis, *Irish Machine Vision and Image Processing Conference (IMVIP)*. **Best Overall Paper Award**

Vincent Andrearczyk and Paul F. Whelan (2015), Dynamic Texture Classification using Combined Co-Occurrence Matrices of Optical Flow, *Irish Machine Vision and Image Processing Conference (IMVIP)*. **Best Overall Paper Award**

Table of contents

List of figures

List of tables

List of abbreviations

AE	Auto-Encoder
ANN	Artificial Neural Networks [1]
BN	Batch Normalisation [2]
BoF	Bag of Features [3]
CNN	Convolutional Neural Network [4, 5]
DT	Dynamic Texture
DT-CNN	Dynamic Texture Convolutional Neural Network
DWT	Discrete Wavelet Transform
FCN	Fully Convolutional Network [6]
FCNT	Fully Convolutional Network for Texture
FV	Fisher Vector [7]
FV-CNN	Fisher Vector Convolutional Neural Network [8]
GAN	Generative Adversarial Network [9]
GLCM	Grey Level Co-occurrence Matrix [10]
GMM	Gaussian Mixture Model
GMRF	Gaussian Markov Random Field
H&E	Hematoxylin/Eosin
IFV	Improved Fisher Vector [11]
K-NN	K-Nearest Neighbours
LBP	Local Binary Pattern [12, 13]
LBP-TOP	Local Binary Pattern on Three Orthogonal Planes [14]
LDS	Linear Dynamical System
LRN	Local Response Normalisation [15]
MLP	Multi Layer Perceptron [1]
MRF	Markov Random Field
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit [16]
RNN	Recurrent Neural Networks [1]
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform [17]

SVM	Support Vector Machine
T-CNN	Texture Convolutional Neural Network
TS-CNN	Texture and Shape CNN
VGG-M	Visual Geometry Group-Medium [18]
VGG-VD	Visual Geometry Group-Very Deep [19]
VLAD	Vector of Locally Aggregated Descriptors [20]

Abstract

Deep learning for texture and dynamic texture analysis

Vincent Andrearczyk

Texture is a fundamental visual cue in computer vision which provides useful information about image regions. Dynamic Texture (DT) extends the analysis of texture to sequences of moving scenes. Classic approaches to texture and DT analysis are based on shallow hand-crafted descriptors including local binary patterns and filter banks. Deep learning and in particular Convolutional Neural Networks (CNNs) have significantly contributed to the field of computer vision in the last decade. These biologically inspired networks trained with powerful algorithms have largely improved the state of the art in various tasks such as digit, object and face recognition. This thesis explores the use of CNNs in texture and DT analysis, replacing classic hand-crafted filters by deep trainable filters. An introduction to deep learning is provided in the thesis as well as a thorough review of texture and DT analysis methods. While CNNs present interesting features for the analysis of textures such as a dense extraction of filter responses trained end to end, the deepest layers used in the decision rules commonly learn to detect large shapes and image layout instead of local texture patterns. A CNN architecture is therefore adapted to textures by using an orderless pooling of intermediate layers to discard the overall shape analysis, resulting in a reduced computational cost and improved accuracy. An application to biomedical texture images is proposed in which large tissue images are tiled and combined in a recognition scheme. An approach is also proposed for DT recognition using the developed CNNs on three orthogonal planes to combine spatial and temporal analysis. Finally, a fully convolutional network is adapted to texture segmentation based on the same idea of discarding the overall shape and by combining local shallow features with larger and deeper features.

Chapter 1

Introduction

Computer vision is a broad research field which deals with the extraction of information and understanding of images and videos using computer algorithms. It encompasses various problems including detection, segmentation, recognition, motion estimation and image restoration. The technological advances and the amount of available data offer an increasingly wide range of applications. The focus of this thesis is the extraction of information from texture images and Dynamic Texture (DT) sequences for recognition and segmentation using deep learning methods.

The rest of this chapter is organised as follows: The notions of texture and DT are introduced in Section 1.1 as well as their analysis by computer vision, including various tasks, applications, challenges, and approaches. The motivation for this work is introduced in Section 1.2 and a summary of the thesis is provided in Section 1.3 including the main contributions. Finally, the outline of the thesis is described in Section 1.4.

1.1 Texture and dynamic texture analysis

1.1.1 What is texture?

In common language, texture generally refers to object surfaces, e.g. rough or wavy variations from a flat surface. Texture can therefore refer to the sense of touch or visual effect of surfaces. Note that texture can alternatively refer to other senses or meanings such as sound texture in music, smell texture of perfumes and texture in text. In this thesis, the term texture (or static texture) refers to the visual texture used in the fields of image processing and computer vision. Visual textures do not solely represent and depend on object surfaces but are rather defined by texture properties of image regions. In biomedical imaging, for instance, textures are not necessarily related to a surface but to the image acquisition of an organ or tissue. Nevertheless, natural texture images reflect some physical variations of an observed scene and

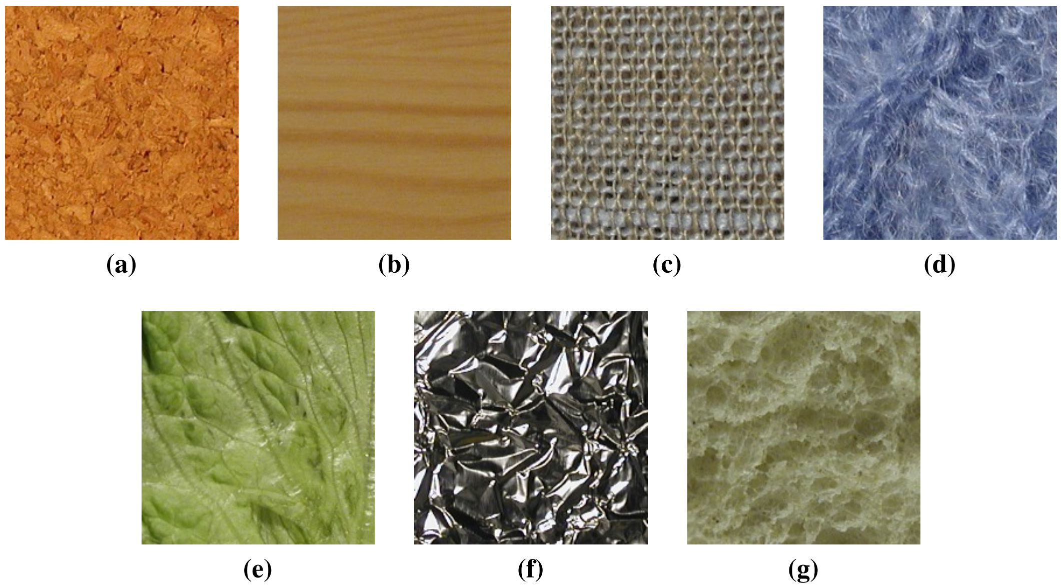


Figure 1.1: Examples of texture images from the kth-tips-2b database [22]. (a) cork, (b) wood, (c) linen, (d) wool, (e) lettuce, (f) aluminum foil and (g) white bread.

reflect the illumination and image acquisition including viewpoint and quality of the camera or other imaging techniques.

In this context, texture, together with colour, is a fundamental visual cue in image processing and computer vision which provides useful information about image regions. Colour commonly refers to the distribution of pixel intensities and can be described, for instance, by histograms of intensities across a region. Texture, on the other hand, is often defined as the spatial variation and arrangement of pixel intensities [10, 21], although there is no generally accepted definition. A texture region obeys some statistical properties and exhibits repeated patterns with some extent of variability in their appearance and relative position [3, 10]. Examples of repeated patterns include wood oriented patterns (see Figure 1.1b) or cells in a biopsy tissue image (see Figure 3.6, page 62). Textures may range from perfectly stochastic (i.e. no repetitivity) to perfectly regular (i.e. exactly the same patterns repeated across the texture region at regular spatial intervals). Textures can be arbitrarily described with terms as simple as oriented lines and spots or more complex semantic properties such as directionality, smoothness, coarseness, density, randomness and regularity.

Surfaces of natural and man-made objects, as well as other natural phenomena, exhibit various types of textures including the examples illustrated in Figure 1.1. Human texture perception has been widely studied, as described in Section 2.2.1, and has largely contributed to the design of texture analysis methods.

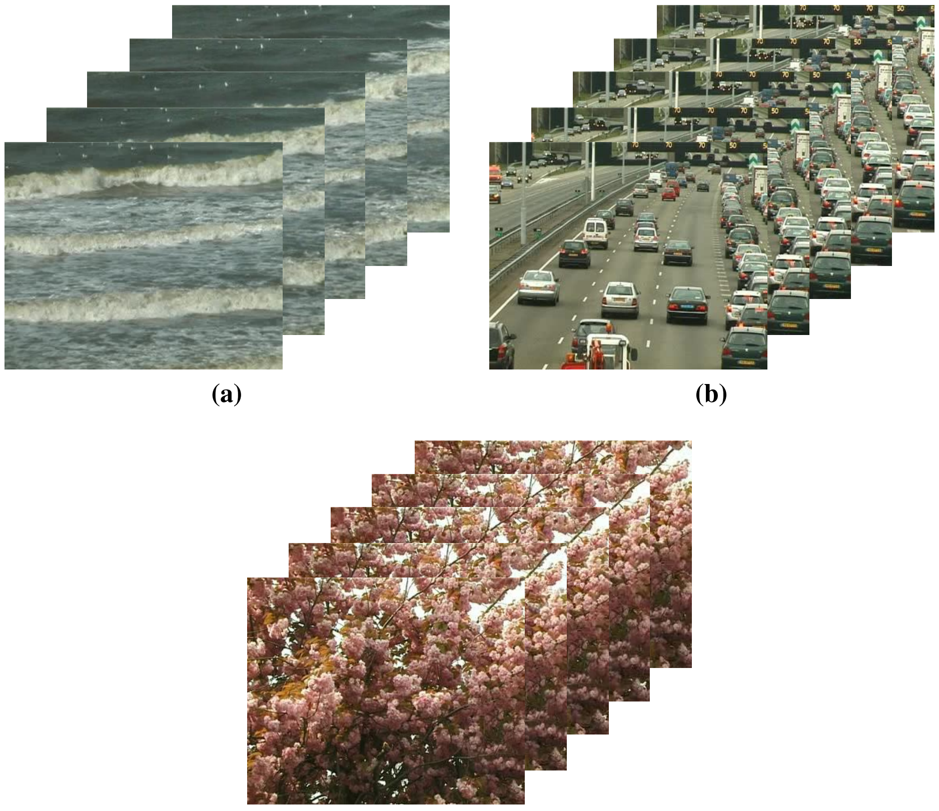
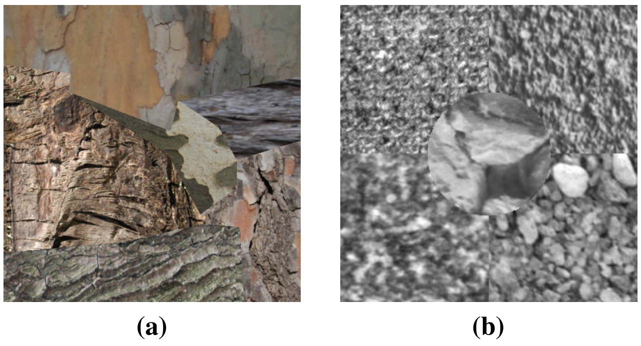


Figure 1.2: Examples of DT sequences from the DynTex database [24] of the following classes: (a) sea, (b) traffic and (c) trees.

1.1.2 What is dynamic texture?

DT, also called temporal texture, is an extension of static texture to the spatiotemporal domain, introducing temporal variations such as motion and deformation. A DT is a sequence of images which exhibits certain spatial repetitivity (similar to spatial textures) as well as stationary properties in time. DT is distinguished from two other types of motion in [23], namely motion event (e.g. opening a door) and activity (e.g. running), both with a compact spatial structure. Similarly to spatial texture properties ranging from stochastic to regular, a large range of temporal properties can describe a DT. It can be described, for instance, by various rigid (e.g. translation and rotation) and non-rigid (e.g. diffusion) transformations as well as temporal periodicity. Examples of natural processes which exhibit DTs include smoke, clouds, trees and waves. Figure 1.2 illustrates three examples of DT sequences typically used in DT recognition.



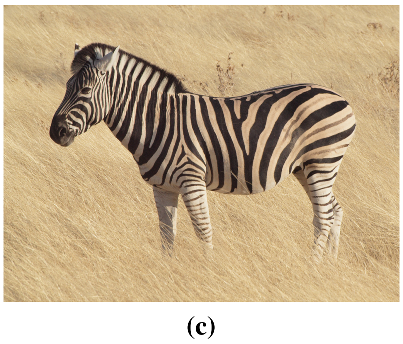


Figure 1.3: Examples of segmentation problems. (a) Image from the Prague texture segmentation benchmark [26]: mosaic with six texture regions, (b) texture mosaic generated from Kylberg texture images [31] including five texture regions, and (c) highly textured zebra skin and grass background.

1.1.3 Analysis and applications

Texture images and DTs can be automatically analysed by computer vision approaches. The analysis of texture embraces several problems including texture classification [13, 25], segmentation [21, 26], synthesis [27, 28] and shape from texture [29, 30]. These tasks involve the development of an algorithm to automatically make a prediction from an unknown image or to synthesise an image. It therefore requires the extraction, from raw pixels, of meaningful features to describe the texture properties. Texture *classification* is the process of assigning an unknown texture image to one of a set of class labels such as “grass”, “wood” and “bricks”. The classification can also be binary such as “malignant” vs. “benign” cancer or texture of interest vs. all other textures (i.e. specialist). The classification of textures is generally used in a supervised approach with multiple training samples for each class. Several texture images typically used in the literature are shown in Figure 1.1. In texture *segmentation*, an image is partitioned into multiple regions of homogeneous texture properties. Examples of texture images to segment are illustrated in Figure 1.3. In unsupervised segmentation, there is no a priori knowledge of the textures present in the image. On the other hand, a supervised model can be trained with example images or information on the textures to segment. Texture classification and segmentation are used in various applications including document processing [32], remote sensing (e.g. satellite imagery [32]), industrial inspection (e.g. paint inspection [32], defect detection [33]), image retrieval [34], and biomedical imaging [32, 35]. Texture *synthesis* refers to the generation of texture images from (generally smaller) texture samples, maintaining identical texture properties. Texture synthesis is frequently used in computer graphics to make surfaces look real or to manipulate images [36, 37], in image compression by storing only a sample of a texture region, or for image inpainting by filling holes in images [32, 38]. Finally, *shape from texture* is used to reconstruct a 3D surface from a 2D image by estimating the shape or orientation based on the texture properties. Typically, the shape and depth of an object can be estimated based on the visual appearance (deformation) of the surface texture resulting from the projection of the 3D object onto the 2D plane [29, 30].

The analysis of DTs is a relatively recent research topic, from the early nineties, as compared to the static texture analysis which started in the fifties. It embraces the same major problems as texture analysis including classification, segmentation, and synthesis. Methods to analyse DTs should capture the spatial and temporal variations of a sequence of images, i.e. static spatial texture properties and dynamics. The analysis of DTs is essential for a large range of applications including remote monitoring and surveillance (e.g. forest fire, traffic, and crowd) [39–41], medical image analysis and remote sensing. The increasing amount of video data due to smartphones, surveillance, medical imaging, robotics, etc., offers an endless potential for applications of DT analysis.

These static and temporal texture problems are reviewed in more details in Section 2.2.3 with a particular attention on classification and segmentation as the major focus of this thesis.

1.1.4 Challenges

The vast majority of natural textures are easily detected, segmented and recognised by humans. The visual system has developed throughout millions of years of evolution to efficiently perceive textures as they carry useful information about an observed scene. Yet, the automatic analysis of textures by computer algorithms remains a challenging problem. The level of abstraction in a computer vision system can be organised in the following list of concepts of increasing abstraction: pixel, image primitive (e.g. edge or spot), texture, region, object, and scene. While the level of abstraction required for texture analysis is relatively low as compared to some object recognition and scene understanding problems, it involves multiple difficulties and challenges.

The major challenge results from the diversity and complexity of natural textures. For instance, considering only wooden surfaces, an extremely wide range of textures can be found with different types of wood, its age, condition and cut as well as illumination and image acquisition variations (e.g. point of view, orientation, noise, and blur). As a result, training samples may be significantly different from test images, requiring a good generalisation of recognition models to avoid overfitting. It is also a source of high intra-class variation, making the definition of a discrimination rule challenging. Moreover, certain approaches are well designed for one type of textures (e.g. regular, oriented, sparse, small or large texture patterns, etc.) but will fail for other types as will be explained in Chapter 2. Developing a method to analyse various types of texture with robust generalisation and multiple invariances is a complex task, as demonstrated by the research conducted over many decades in this field. These difficulties are also emphasised by the variety of definitions given to visual textures which largely depends on the application and type of images [32].

An important notion which requires particular attention in texture is the scale, closely related to the spatial frequency. The perception or analysis of textures highly depends on the scale at which a surface or scene is viewed. Leaves on a tree, for instance, exhibit a certain texture. A single leaf exhibits another texture such as parallel lines (veins) on both sides of the centre vein. Zooming in even closer, patterns formed by smaller (higher order) veins may exhibit again a different texture. Repetitive texture patterns can therefore be found at multiple scales and frequencies. Texture descriptors generally require scale invariance as textures acquired from different viewpoints should be recognised as the same class. On the other hand, the scale may be a discriminative information in some applications with a fixed viewpoint. Scale variance can even emerge from a single image, for instance with a non-flat surface, a camera angle distant from the surface normal or a “fisheye” effect as shown in Figure 1.4. Multi-scale analysis is therefore necessary and most applications require local and/or global scale invariance. A pyramid representation is commonly used in texture analysis to perform a multi-scale analysis by the successive smoothing and downsampling of an image. Additionally to the spatial scale, DTs may contain meaningful information at multiple scales and frequencies in the temporal domain. Besides the scale, natural textures may vary in terms of orientation, illumination, acquisition noise, occlusion, clutter and other visual appearances. Many applications therefore require various types of invariances to be able to correctly classify and segment textures. Another difficulty in texture analysis is the extraction of sufficient discriminative information while maintaining reasonably low dimensionality and low redundancy of the texture descriptors. Finally, the lack of information about the textures and the number of regions in unsupervised segmentation is a challenging problem.

Regarding temporal textures, the human visual system is extremely efficient and accurate at evaluating the motion and appearance of a scene to effortlessly

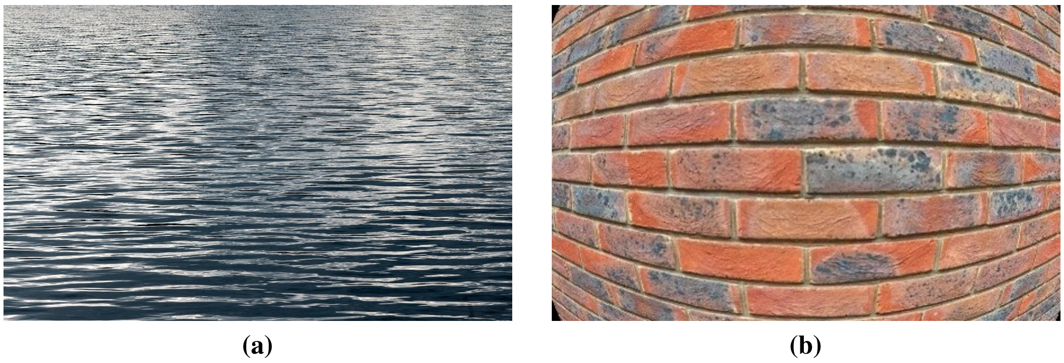


Figure 1.4: A texture distortion resulting in patterns of varying sizes, shapes and frequency. (a) Water ripples appear at different scales and frequencies due to the camera angle to the surface normal, and (b) Fisheye effect on bricks texture due to image acquisition.

recognise DTs. Yet, it is a difficult computer vision task. Adding to the challenges of static textures described previously, a major difficulty arises from the wide range of dynamics originating from complex motions and interactions of objects or particles as well as camera motion. The temporal variation of a DT, however, provides a valuable additional information as compared to the static texture when used in a recognition scheme. Therefore, the discrimination of DTs should greatly benefit from a spatiotemporal analysis. In this context, one major challenge is to optimally extract and combine spatial and temporal information, as detailed in Chapter 2.

1.1.5 Classic approaches

Most texture analysis methods involve the extraction of features which describe the properties of a texture in order to recognise it, segment it or synthesise similar samples. Deriving features is necessary to extract meaningful information from the large number of pixels in an image. Indeed, pixels in their raw form are not descriptive enough and of too high dimensionality to enable a discrimination of texture regions or images. Various feature extraction methods have been developed in the last decades, partly inspired by the studies of the human and animal visual systems [42–44]. Texture features should be informative, non-redundant and should offer certain invariances required for a given application. Many classic early texture analysis approaches use local descriptors in the form of binary patterns [13] or filters [21] to extract local or global features. Local descriptors can be encoded into a global descriptor for an entire image or region, for instance by a histogram of occurrence. These descriptors can then be segmented or classified using classic machine learning methods such as Support Vector Machine (SVM). These approaches use hand-crafted and pre-defined local descriptors which have been outperformed by descriptors learned from the training samples such as Bag of Features (BoF) [3] and Fisher Vector (FV) [7].

The first DT studies were built upon the extensive work previously carried on static textures as well as knowledge of physics targeting specific dynamics [45]. DT analysis methods are, for the most part, extensions of classic texture methods to the spatiotemporal domain. For instance, motion and appearance information can be extracted and combined [46] or a sequence can be considered as a volume to extract 3D local descriptors [14] or filter responses [47].

A few recent approaches for static and temporal texture analysis combine the learning of features together with their classification, segmentation or synthesis within a deep neural network. The focus of this thesis is on exploring these deep learning methods and adapting convolutional networks to the analysis of texture. More details on hand-crafted, learned, shallow[1], and deep descriptors are provided in the literature review (Chapter 2), while deep learning is introduced in Appendix A.

1.2 Motivation for the thesis

As mentioned in Section 1.1.3, the analysis of static and temporal textures is crucial in many applications. Currently, biomedical imaging may be the field which benefits the most from it. As detailed in [35], the analysis of texture is relevant in various applications of biomedical imaging including the detection of lesions, nodules and mitoses, the characterisation of tumors, cancers and disease tissues, as well as image retrieval and radiomics. These analyses are conducted on various image modalities including Magnetic Resonance Imaging (MRI), Computed Tomography (CT) scans, Positron Emission Tomography (PET), ultrasounds and biopsies. Besides their application in biomedical imaging, texture and DT analyses are used in various areas including remote sensing, image retrieval, and industrial inspection.

Classic shallow hand-crafted features lack invariances and abstraction for many applications and must be specifically designed for particular problems. In turn, these methods do not generalise well to complex and numerous textures with high intra-class variation as encountered in various texture analysis problems. Their use is therefore limited and their performance often dependent on the application.

Deep learning is a biologically inspired machine learning approach which trains deep Artificial Neural Networks (ANNs) to perform complex tasks. A Convolutional Neural Network (CNN) [4, 5] is a deep neural network developed specifically for grid-like data such as images and videos. The success of deep learning in computer vision is evident in the last decade as shown by the state of the art in various applications,

[1]The term "shallow" refers to classic feature extraction methods which do not use deep neural networks, as well as shallow networks (generally less than three layers). Shallow descriptors can be hand-crafted (e.g. filter banks) or learned (e.g. dictionary learning).

the number of studies and the media exposure. Neural networks, deep learning, and CNNs are introduced in Appendix A. CNNs can successfully learn invariances required for various texture analysis tasks, e.g. to rotation, translation, and scale. Moreover, CNNs are designed in a succession of convolutions with trainable filters, combined into increasingly complex non-linear banks of filters. This filter bank design is well suited to texture analysis and shares similarities with classic feature extraction methods, while replacing hand-crafted features and standard classifiers by powerful trainable kernels and end-to-end training. The first layer of a trained CNN is similar to filter banks used in texture analysis [48] with oriented edges and other simple patterns. Deeper features can be thought of as a more complex learned non-linear filter bank.

Despite this appropriate design, complex and large structures and shapes emerge in deep layers rather than simple texture descriptors as demonstrated in [49]. For example, in the VGG-16 network [19] trained on ImageNet [50], the neurons in the last convolution layer respond to complex shapes in the input image such as faces, cars, persons, etc. Moreover, the receptive field (see Appendix A.4.3) grows throughout the network, therefore neurons in the deepest layers analyse large portions of the input image or even the entire image. This hierarchical feature learning of growing complexity (e.g. pixels $\rightarrow$ edges, blobs $\rightarrow$ wheel, door $\rightarrow$ car, truck) is required for object recognition. Textures, however, are better characterised by the distribution of small descriptors of limited complexity.

These observations motivate the study of CNNs applied to texture and DT analysis as proposed in this thesis, with a key idea of discarding the overall shape analysis to focus on the orderless pooling of filter responses. Orderless pooling refers to the computation of a global descriptor from local descriptors (filter responses) regardless of their spatial location in the input image or in the feature map. The domain transferability of pre-trained filters should also allow to pre-train networks on very large datasets (e.g. ImageNet) to transfer and finetune the parameters on small texture datasets.

1.3 Thesis summary and main contributions

The aim of the research outlined in this thesis is to develop deep learning methods adapted for the analysis of texture images and DT sequences. The thesis is split into three research problems: texture classification, DT classification, and texture segmentation. The ideas and analyses common to these three parts include discarding the overall shape analysis of CNNs, reducing the number of trainable parameters, evaluating and using the domain transferability of the latter and visualising and interpreting learned features. The goal is to improve the accuracy and reduce the complexity compared to existing networks, as well as gaining an insight into how and what deep convolutional networks learn from texture and DT datasets.

The major contributions of the thesis can be summarised as follows:
(1) A CNN architecture is developed for the classification of textures, discarding the overall shape analysis by orderless pooling of filter responses [25];
(2) a framework is introduced for an application to biomedical tissue images classification [35, 51];
(3) a method fusing texture-specific CNNs on spatial and temporal slices of sequences is proposed for DT recognition [52];
(4) a fully convolutional architecture is developed for the segmentation of texture regions and used in supervised and unsupervised tasks [53];
(5) several analyses are conducted to get an insight into what and how CNNs learn to recognise and segment images including domain transferability, depth analysis and visualisation of learned features [25, 35, 52, 53].

1.4 Outline of the dissertation

The literature review of texture and DT analysis is presented in Chapter 2 with a focus on classification and segmentation problems. Studies of human texture perception and early feature extraction approaches are introduced as a basis on which more recent trends are built. Various texture feature extraction methods are described including structural, statistical, spectral, local descriptors and bag of features. The use of these features in texture analyses is described, with an emphasis on classification and segmentation.

Chapter 3 presents a CNN architecture specifically designed for texture classification. Part of this work was published in [25, 35, 51]. Networks are developed, based on existing architectures, to discard the overall shape analysis by orderless pooling of intermediate convolution layers. The idea is that intermediate layers extract texture patterns densely across the feature maps and can be pooled by calculating their average responses. The developed method is tested on several texture classification datasets and compared with the state of the art. Higher accuracy is obtained with a lower computational complexity as compared to classic CNNs. The domain transferability of pre-trained networks is analysed as well as the features learned by the networks using visualisation techniques (see Appendix A.4.7). An alternative approach, combining texture and global shape analysis within a single network, is also proposed. Finally, an application of the developed texture specific network is presented on the classification of biomedical tissue images. Making use of the homogeneity and repetitivity of the tissue textures and of the size of the images, the latter are split into a grid of subimages which are used in an ensemble classification. The developed method significantly outperforms the state of the art on several benchmarks evaluating the recognition, among others, of malignant lymphomas.

The developed texture-specific CNN is applied to the recognition of DTs in Chapter 4. Part of this work was proposed in [52]. The analysis of spatial and temporal slices regularly sampled along the three axes is permitted by the repetitivity property of DTs in space and time. A late fusion of predictions on three orthogonal planes is proposed. This method combining spatial and temporal analysis outperforms the state of the art on multiple DT classification benchmarks. Various analyses are conducted including the contribution of each plane and the domain transferability of trained parameters.

A new deep learning approach for texture segmentation is proposed in Chapter 5 and was introduced in [53]. Sharing the idea of discarding the overall shape analysis with the preceding sections, a Fully Convolutional Network (FCN) is adapted for texture images. Using "skip" layers, filter responses are combined at multiple scales as the so-called "where" (local details) and "what" (more complex and larger features) information. Due to the homogeneity of texture properties across regions, the developed network can be trained on non-segmented images, i.e. classic texture classification datasets. It is also shown that this network can be trained on very little training data by relying on the repetitivity of texture patterns. This allows developing a supervised framework by training on a single image per class as well as an unsupervised framework by training on patches of the test image itself. The proposed method significantly outperforms the state of the art on multiple texture segmentation benchmarks.

A conclusion is provided in Chapter 6 including discussions, limitation, and suggestions for future work.

Finally, a technical introduction to neural networks, deep learning and CNNs can be found in Appendix A. Major concepts, architectures and training methods are introduced and will be referred to throughout the thesis. In particular, artificial neurons, ANNs, backpropagation, gradient descent, CNN building blocks and recent architectures are presented.

Chapter 2

Literature review

2.1 Introduction

This chapter reviews various major advances in the fields of texture analysis (Section 2.2) and DT analysis (Section 2.3). It includes the extraction of features which describe spatial or spatiotemporal texture regions or images as well as their use in classification, segmentation, and other analyses. Recent deep learning methods used in texture and DT analysis are also presented. An introduction to neural networks, deep learning and CNNs can be found in Appendix A.

2.2 Texture analysis

The analysis of texture is traditionally divided into four problems: classification, segmentation, synthesis, and shape from texture. A key processing step shared by most texture analysis methods is the extraction of features which describe the textures. Feature extraction methods are therefore explained followed by classification, segmentation, and other analyses which use these texture features including K-Nearest Neighbours (K-NN) and SVM.

This section is organised as follows: The human perception of textures is introduced in Section 2.2.1. Various shallow feature extraction methods are described in Section 2.2.2. Texture classification, segmentation, synthesis, and shape from texture, using the described shallow descriptors, are introduced in Section 2.2.3. Finally, recent work on deep CNN features and deep learning methods in texture analysis are detailed in Section 2.2.4.

2.2.1 Texture perception

The perception of texture is crucial for humans as every object surface reflects a particular texture which enables, among other analyses, to estimate its shape or tactile perception and is a basis for further estimation of depth, motion and object recognition. The human perception of texture has been widely studied [42] and has largely influenced the development of computer-based texture analysis methods. Julesz's first conjecture [42] stated that two textures with identical second-order statistics (based on pairs of pixel values) cannot be discriminated by the preattentive textural system. He later rejected this conjecture with counter-examples and proposed a "theory of textons" which assumes that the preattentive discrimination of texture regions is based on similarity/dissimilarity of textons. Textons are described in [43] as particular local texture features such as corners, end-lines, and closures. In other words, texture regions with identical texton information are not preattentively discriminable. The theory of textons has inspired many texture analyses including early structural methods and more recent dictionary learning ones explained in Section 2.2.2. Studies in [44] have also shown that simple cells of the visual system perform a frequency and orientation analysis which has greatly motivated spectral filtering methods such as Gabor filter banks [21, 32].

2.2.2 Classic texture feature extraction

Traditionally, the extraction of features is necessary for all texture analysis methods, as a starting point to extract meaningful information from the raw pixel values. Many feature extraction methods have been developed in the last decades. Various methods are described in the following sections. The extracted features can then be clustered, classified etc. to solve texture analysis problems as explained in Section 2.2.3. Despite an inevitable overlap of concepts in these approaches, they are grouped to provide a readable and structured review. In this review, the methods are grouped into structural, model-based, statistical, spectral, local descriptors, learned visual dictionaries and deep learning approaches.

Structural

Structural approaches consider textures as a composition of texture primitives regularly arranged according to some spatial organisation rules. These methods first involve the identification of the texture primitives, followed by an inference of the placement rules or a statistical description of the primitives shapes. Texture primitives generally refer to blobs, i.e. regions in the image with uniform grey levels [32], as these are considered meaningful basic elements of a texture. This approach was supported by psychophysical experiments which have demonstrated that humans can strongly perceive the structural properties of regular textures [42]. Several methods have been developed to identify blobs including mathematical morphology [54], boundaries detection such as Laplacian of Gaussian (LoG) or Difference of Gaussian

(DoG) filters [55–57], and region split and merge followed with blobs approximation by medial axes [58]. Other texture primitives include edges [59] and skeleton extracted by histogram analysis [33].

Once the primitives are identified, texture descriptors can be computed by inferring placement rules which define the spatial relationships between the primitives. Generalised co-occurrence matrices [59] describe second-order statistics of edges, inspired by the Grey Level Co-occurrence Matrix (GLCM) and Haralick features [10, 60] introduced later. The Voronoi tessellation method [57] represents each primitive by a single point (e.g. centroid), forming a set of points S. The perpendicular bisector of the line joining a pair of points $(P,Q) \in S$ is constructed, splitting the image into two regions (pixels closer to P, and those closer to Q). Repeating this process for all the pairs of points in S, a polygonal region is obtained for each point $P \in S$ (i.e. for each primitive). Such region contains all the pixels that are closer to its centre point P than to all other points. The shapes of the polygons which represent the spatial organisation of the primitives can be used to derive texture features.

Alternatively to the placement rules, measures and statistics of homogeneous primitives can be computed including intensity, shape and orientation [61, 58].

By considering primitives and placement rules, structural approaches are generally only used for regular textures. These methods are, by definition, not designed for textures with a high degree of randomness and variability of patterns as encountered in recent texture datasets and in real life images.

Model-based

In a model-based approach, the fundamental qualities of a texture are captured by a model with estimated parameters. These parameters can be used as texture features or to synthesise textures of desired properties. The most popular model-based approaches include Markov models and fractals as described in the following text.

Markov models are based on the Markov property which assumes that the future state of a system depends only on the current state. This translates to images by local dependencies of pixel intensities. A Markov Random Field (MRF) assumes that a pixel intensity depends only on its neighbouring pixels, capturing local contextual constraints to globally model an image [62]. To be more precise, given its neighbouring intensities, a pixel is conditionally independent of other pixels in the image. A MRF is a graphical model which forms an undirected graph with pixels or superpixels as random variables and with edges only between neighbouring pixels/superpixels. The parameters of the model are estimated to best fit the image based on an optimisation method that minimises an energy function. The estimated parameters are commonly used as texture features; yet, the determination of the energy function and its optimisation are difficult. Gaussian MRFs (GMRFs) have been extensively used in texture analysis [63]. A GMRF model is expressed by the probability of a pixel at position (x,y) having a value I_{xy} given the intensities in its neighbourhood N_{xy}. This probability is calculated as follows:

$$p(I_{xy}|I(i,j),(i,j) \in N_{xy}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ \frac{\left(I(x,y) - \sum_{l=1}^{n} \alpha_l s_{xy;l}\right)^2}{2\sigma^2} \right\} \tag{2.1}$$

where α_l weights the neighbours' influence, $s_{xy;l}$ is a sum of pairs of pixels in N_{xy} symmetric w.r.t. (x,y) and n is the number of such pairs. The latter depends on the size N_{xy} and relates to the order of the GMRF model. A first-order model only considers pairs of direct 4-connected neighbours, a second-order model considers 8-connected neighbours, etc. The parameters $\boldsymbol{\alpha}$ and σ (standard deviation) can be estimated by least square error on the entire texture image or region. Wold decomposition [34] is another model-based method which measures periodicity, directionality, and randomness by the decomposition of a homogeneous random field into three mutually orthogonal subfields. Other popular models used in texture analysis include the Simultaneous Auto-Regressive (SAR) [64] which is an instance of MRF and the multi-resolution SAR [65]. Major difficulties with random field methods include finding an appropriate energy function and optimising it.

Fractal models propose another popular approach to model images [66, 67]. The fractal geometry is based on the concept of self-similarity across scales, i.e. repeated patterns at multiple scales. The fractal dimension D is a measure of this fractal repetitivity and is related to the perceived roughness of a texture. It is calculated as follows:

$$D = \frac{\log(N)}{\log(\frac{1}{r})} \tag{2.2}$$

where N is the number of repeated patterns downsampled by a ratio r. Several methods have been proposed to estimate the fractal dimension including the box-counting and differential box-counting methods [68, 69], fractional Brownian motion with spectral analysis [70], and area-based approaches such as the triangular prism method [71]. A major drawback of fractal approaches is that the fractal dimension does not capture sufficient textural information since textures with different appearances can have similar fractal dimensions [68]. Another measure is commonly used to discriminate such textures, namely lacunarity [66] which measures the sparsity of the texture by considering how fractals fill the image, i.e. gaps between the fractal patterns.

Statistical

Statistical approaches describe the relationships between pixel values based on first-, second-, or higher-order statistics.

First-order features provide information on the values of individual pixels, but not on the relative positions of the pixel values. These include mean, variance, skewness, and kurtosis.

Second- and higher-order statistics are commonly extracted by matrix-based approaches. Second-order statistics consider the distribution of pairs of pixel values, typically derived with the GLCM [10, 60].

The GLCM aims at describing the relationships between neighbouring pixel intensities by analysing their joint probability function. It summarises the occurrence of pairs of pixels (horizontally, vertically or diagonally) in an image. The $(i, j)^{th}$ entry of the matrix $p(i, j)$ represents the number of occurrences of a pixel with (quantised) intensity value i, separated from another pixel with intensity value j at a distance d in the direction θ. An example of GLCM computation with a small texture region is illustrated in Figure 2.1. GLCMs contain meaningful second-order statistics

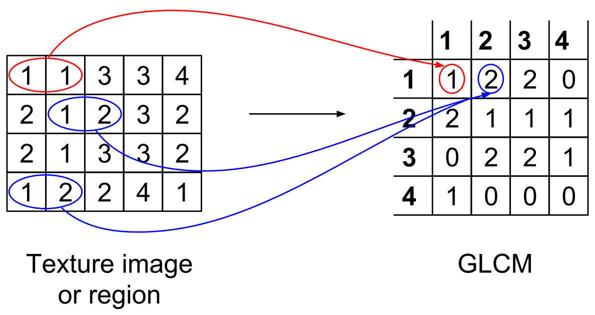


Figure 2.1: An example of GLCM computation with a small texture region and four grey levels [10]. The distance between neighbours is $d = 1$ and the orientation is $\theta = 0$.

information which can be extracted (possibly averaged for different distances d and directions θ) by a set of features including correlation, contrast, homogeneity (angular second moment) and dissimilarity (inverse difference moment). These features are referred to as the Haralick texture features [60]. The contrast measure, for instance, is computed as follows:

$$Contrast = \sum_{i,j} p(i, j)(i - j)^2 \tag{2.3}$$

where i, j run over the rows and columns respectively. The major drawbacks of GLCMs are the high dimensionality of the matrix if used raw (without the computation of features) and the high correlation of the Haralick features.

Higher-order statistics analyse the joint distribution of more than two pixels. Instead of the occurrence of pairs of pixels, the Grey Level Run Length Matrix (GLRLM) [72] summarises the occurrence of runs of pixels, i.e. the occurrence of a given grey value in a given direction. Higher-order statistical features can be extracted from the GLRLM. The Grey Level Size Zone Matrix (GLSZM) [73] is similar to the GLRLM except that zones of connected pixels are considered instead of oriented runs of pixels. Therefore, it does not require the computation in several directions contrary to GLCM and CLSZM. Features similar to the GLCM features can be extracted from the matrix to statistically describe a texture.

Note that the described matrices require a quantisation of the pixel values to limit the size of the matrices and improve robustness to noise and small intensity variations. The best quantisation and directions (for GLCM and GLRLM) may depend on the application. Although improvements of these methods have been proposed, the performance of statistical approaches remains relatively poor on natural texture images.

The autocorrelation function is another statistical approach which computes the dot product of an image with shifted instances of the same image. It has been used to detect repetitive texture patterns (primitives) and describe the regularity and coarseness of textures [74]. Note that the autocorrelation is a signal processing method strongly related to the Fourier transform described below.

Spectral analysis

Signal processing approaches include filter banks, wavelets and Fourier transforms. These methods analyse the frequency (or spatial-frequency) content of textures in the spatial domain only (e.g. edges, Laws features, steerable filters), in the frequency domain (e.g. Fourier transform), or in both frequency and spatial domains (e.g. Gabor filters and wavelet transforms). A spatial filtering is an image operation which computes a function of the intensities in the local neighbourhood of each pixel.

Spectral methods in the spatial domain convolve the texture images with spatial filters, extracting frequency information by measuring the variations in local neighbourhoods. This is typically computed by convolving the image with small filters or kernels (see Appendix A.4.3) resulting in a set of images being the filters responses. Texture features are commonly based on statistics of filter responses. Edge-based filters, e.g. Robert's or Sobel's [75], attempt to describe the coarseness of textures based on the density of edges. Laws [76] developed a set of nine 5×5 filters to extract local responses to meaningful patterns. The filters are computed as the product of pairs of vectors from a set of four vectors representing a $1D$ level, edge, spot, and ripple respectively. The averaged response to the filters and combination of filters (e.g. to consider horizontal and vertical edges) are used as texture features.

Steerable filters were designed in [77] for the analysis of oriented textures. Steerable filters are a set of orientation-selective filters, obtained at any orientation by a linear combination of basis filters (e.g. derivative of Gaussian at 0° and 90°). Finally, local linear transforms, similar to a bank of FIR (Finite Impulse Response) filters, were used in [78].

Spectral analysis in the frequency domain is performed by Fourier transform. A 2D discrete Fourier transform decomposes an image into its frequency components as a sum of orthogonal basis functions as follows:

$$I(x,y) \overset{\mathscr{F}}{\rightarrow} F(f_x, f_y) = \sum_{x,y} I(x,y) e^{-j2\pi(f_x x + f_y y)} \tag{2.4}$$

where f_x and f_y are the horizontal and vertical frequencies respectively. This complex number can be represented by its real and imaginary parts, or decomposed into the magnitude and phase as follows:

$$F(f_x, f_y) = |F(f_x, f_y)| e^{j\measuredangle F(f_x, f_y)} \tag{2.5}$$

An example of Fourier transform is shown in Figure 2.2. Spatial edges typically exhibit a low frequency in one direction and multiple frequencies in the orthogonal direction. This is represented by straight lines in the Fourier domain (first row in Figure 2.2). Low-frequency components in the centre of the Fourier transform occur from the various flat regions as shown in the second row. Low-frequency components are filtered out in the third row, maintaining high-frequency information, i.e. mainly edges. Note that the Fourier transform for the zero frequencies ($f_x = f_y = 0$) computes the mean of the image which is often considered a colour measure, rather than texture. The Fourier transform can be thought of as representing an image as a weighted combination of vertical and horizontal sinusoids of various frequencies (f_x, f_y), similar to various frequencies in different directions. Note that a convolution in the spatial domain of an image $I(x,y)$ with a filter $w(x,y)$ is equivalent to a multiplication in the Fourier domain of the respective Fourier transforms $F(f_x, f_y)$ and $W(f_x, f_y)$.

$$\begin{aligned} I(x,y) &\overset{\mathscr{F}}{\rightarrow} F(f_x, f_y) \\ w(x,y) &\overset{\mathscr{F}}{\rightarrow} W(f_x, f_y) \\ I(x,y) * w(x,y) &\overset{\mathscr{F}}{\rightarrow} F(f_x, f_y) W(f_x, f_y) \end{aligned} \tag{2.6}$$

Thus a multiplication in the Fourier domain by a band-pass filter is equivalent to a convolution in the spatial domain. A set of (localised) band-pass filters in the Fourier domain can precisely describe the frequencies in the image. This approach, however,

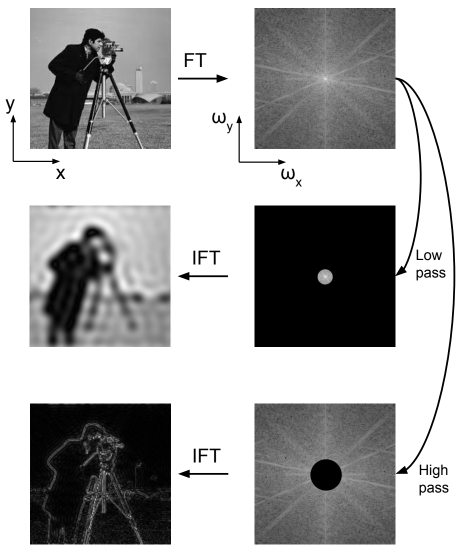


Figure 2.2: An example of Fourier transforms and inverse Fourier transforms (after low-pass and high-pass filters in the frequency domain). Left: spatial domain; right: frequency domain (magnitude of Fourier spectrum).

is not localised in space as a narrow frequency band-pass describes a large spatial region. Thus Fourier transform methods cannot describe local variations of textures. Ideally, a texture analysis should be localised in both spatial and frequency domains, meaning that local spatial properties are analysed together with the frequency components of the image. Yet, the localisation of a texture operator in both domains is limited due to the principle referred to as Heisenberg's uncertainty. It states that many frequencies are required to describe a small spatial region (short pulse) and vice versa since the localisations in both domains are limited by a lower bound on their product:

$$\Delta x \Delta y \Delta f_x \Delta f_y \geq \left(\frac{1}{4\pi}\right)^2 \tag{2.7}$$

Similarly, a descriptor localised in the spatial domain can precisely describe local structures; yet, its localisation in the frequency domain is limited.

A Short-Time Fourier Transform (STFT) can be used to extract both the spatial and frequency information of an image by computing Fourier transforms on local neighbourhoods [79].

Another type of approaches to describe spatial and frequency information is to perform a multi-resolution analysis with a bank of filters of various scales, i.e. the size of the local neighbourhood or window. Small scales extract high-frequency content while large scales extract low-frequency information. Typical multi-scale methods include Gabor filters and wavelet transforms. A Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave. It is typically a band-pass filter with tunable central frequency, orientation and bandwidth. The Gabor form was shown to achieve the theoretical limit of localisation in time (extended later to space) and frequency, ruled by the uncertainty principle (Equation 2.7). It was also shown that it operates similarly to simple cells in the visual cortex of the human visual system [44]. A 2D Gabor filter can be computed as follows:

$$G_{\sigma,f_0,\phi}(x,y) = \exp\left(-\frac{1}{2}\left[\frac{x'^2}{\sigma_x^2} + \frac{y'^2}{\sigma_y^2}\right]\right) cos(2\pi f_0 x' + \phi) \tag{2.8}$$

where σ_x and σ_y are the scales of the Gaussian envelop (standard deviations) in the x and y directions respectively, and $\sigma = \sigma_x = \sigma_y$ in this section for simplicity. f_0 and ϕ are respectively the frequency and phase of the sine wave. x', y' perform a rigid rotation θ of the spatial plane as follows:

$$\begin{aligned} x' &= x\cos\theta + y\sin\theta \\ y' &= -x\sin\theta + y\cos\theta \end{aligned} \tag{2.9}$$

The bandwidth of a filter can be derived from the central frequency and scale:

$$B = log_2\left(\frac{\pi\sigma f_0 + \sqrt{\frac{\ln(2)}{2}}}{\pi\sigma f_0 - \sqrt{\frac{\ln(2)}{2}}}\right) \tag{2.10}$$

Examples of Gabor filters with different orientations, scales and central frequencies are illustrated in Figure 2.3.

A bank of Gabor filters with different frequencies and orientations is commonly used to extract texture features [80, 21]. The Fourier transform of a Gabor function is composed of two Gaussians centred at f_0 and $-f_0$ respectively, of bandwidth inversely proportional to the spatial variance and rotated by the orientation θ. A typical bank of Gabor filters spans the frequency domain as shown in Figure 2.4. A Gabor filter bank can perform a robust multi-resolution decomposition due to its localisation both in the spatial and frequency domain. Other filter banks perform multi-resolution frequency analysis including the Leung-Malik (LM), Schmidt (S) and Maximum Response (MR) sets. The LM set contains second derivatives of Gaussians at 6 orientations, 3 scales and 2 phases (i.e. even and odd symmetry) as well as 8 LoG filters and 4 Gaussians for a total of 48 non-rotation invariant filters. The S set is rotation invariant and contains 13 isotropic Gabor-like filters. The MR sets include the MR4 and MR8 sets. They contain LoG and Gaussian filters (both isotropic) as well as an edge and a bar filter which are reduced to 4 and 8 responses respectively by keeping only the maximum responses. These sets are rotation invariant while considering the angle of maximum response. Thus, unlike

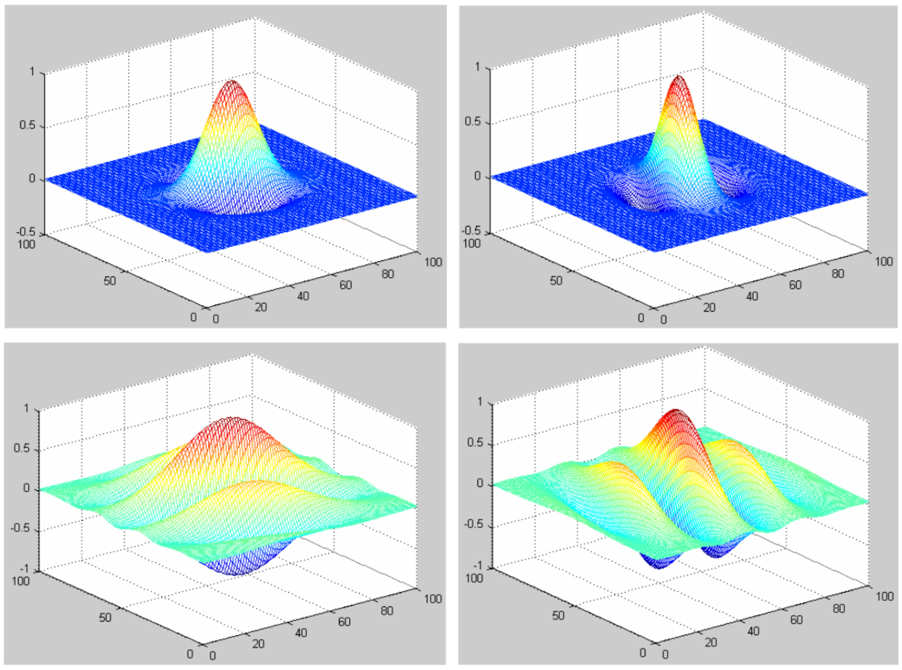


Figure 2.3: 2D Gabor filters for 30°(left) and 120°(right) orientations. Top row: scale $\sigma_x = \sigma_y = 1.0$, central frequency $f_0 = 1.5/2\pi$; bottom row: scale $\sigma_x = \sigma_y = 2.0$, central frequency $f_0 = 2.5/2\pi$.

the S set, co-occurrence statistics can be computed on the orientations which is useful to discriminate isotropic from anisotropic textures.

A wavelet transform is similar to a Fourier transform using a critically sampled bank of local filters (wavelets). While a Fourier transform approximates an image by a sum of sinusoidal plane waves of varying frequencies (Equation 2.4), a wavelet transform approximates the image by a sum of dilated and translated local wavelets. The advantage of wavelet transforms over a Fourier transform, besides a lower computational complexity, is the spatial resolution of the wavelets, as opposed to the sinusoidal basis which discards spatial information. The wavelet transform minimises the Heisenberg uncertainty, capturing localised spatial and frequency information. A Discrete Wavelet Transform (DWT) is performed by applying a series of scaled filters, obtained from a cascade of high-pass and low-pass decompositions followed by downsampling as illustrated in Figure 2.5. Note that the high-pass and low-pass filters represented by a single block in Figure 2.5 separately filter the image horizontally and vertically, as shown in Figure 2.6 for a single scale. The coefficients therefore contain vertical (i.e. rows low-pass and columns high-pass), horizontal (rows high-pass and columns low-pass) and diagonal (high-pass on both rows and columns) details as depicted in Figure 2.7. Each subimage in Figure 2.7 contains information for a specific scale and orientation while maintaining the spatial information. The obtained subimages are commonly used to extract scale dependent

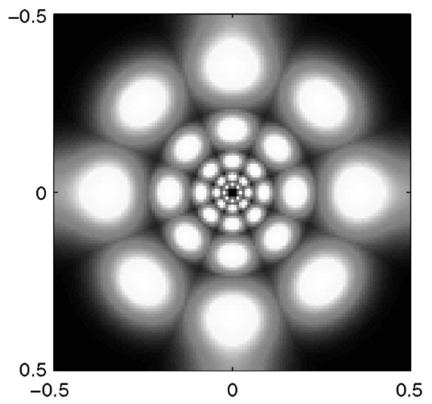


Figure 2.4: A Gabor filter bank in the frequency domain. The set is composed of filters at five scales and four orientations with a total of 20 filters, each resulting in a centre-symmetric pair of lobes. The axes are in normalised frequencies. Figure reproduced from [48].

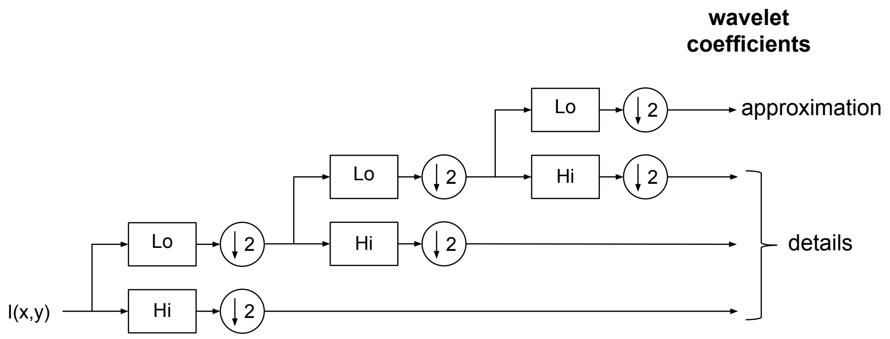


Figure 2.5: A three-level DWT filter bank computing a cascade of high-pass and low-pass filters followed by downsampling, successively decomposing the image into multiple frequency components.

texture features such as energy, variance, and entropy.

The described DWT is derived from the projection of the image onto a basis of wavelet spaces in the horizontal and vertical axes. In DWT, 1D (child) wavelets are obtained by dilation by a power of two and translation of a mother wavelet $\psi(t)$ as follows:

$$\psi_{j,k}(t) = \frac{1}{\sqrt{2^j}}\psi\left(\frac{t-k2^j}{2^j}\right) \tag{2.11}$$

where j and k are the scale and shift parameters respectively. Typical wavelet functions used with DWT include Haar wavelets and Daubechies wavelets [82]. Note that Gabor wavelets can be derived from a mother Gabor filter. Yet, a bank of selected Gabor filters is generally preferred over Gabor wavelets due to their non-orthogonality inducing redundant information in the filtered images resulting from the wavelet decomposition. Another wavelet transform was introduced in [83], called discrete wavelet packet transform. It decomposes not only the high-passed

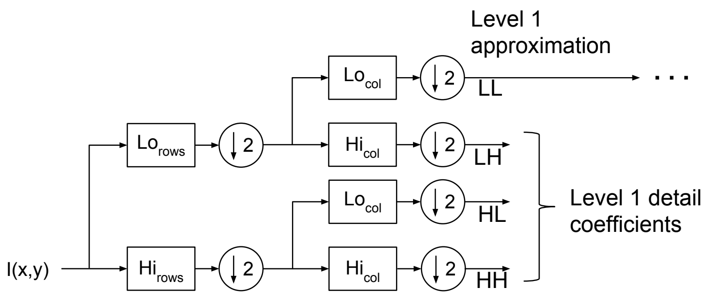


Figure 2.6: A 2D DWT in more detail with high-pass and low-pass filters applied separately horizontally (Hi_{rows}, Lo_{rows}) and vertically (Hi_{col}, Lo_{col}).

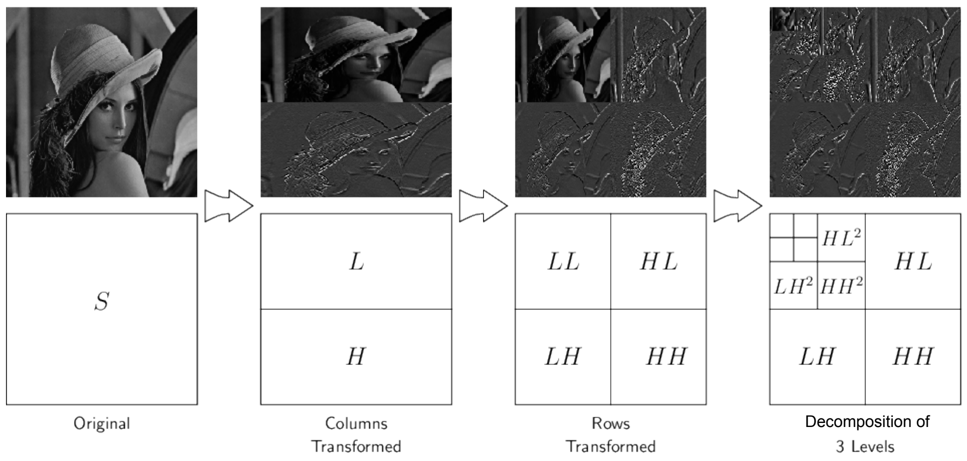


Figure 2.7: A Three-level DWT pyramidal decomposition of an image. Figure reproduced from [81].

approximation images but also the detail images (i.e. HL, LH and HH etc. in Figure 2.7). The major drawbacks of the DWT compared to the Fourier transform include oscillations, shift variance, lack of directionality and aliasing. The Dual-Tree Complex Wavelet Transform (DT-CWT) was developed in [84] to address these problems by using complex wavelets. Inspired by the Fourier transform using real cosine and imaginary sine, one real and one imaginary wavelets are used with a 90° phase difference. Other transforms used in texture analysis include Discrete Cosine Transform (DCT) [38], contourlet transform, and curvelet transform [85].

The filter banks described so far are heuristically designed and generally result in a large computational complexity and high-dimensional feature vectors. Several methods have been developed to optimise the filter banks including eigenfilters, prediction error filters, optimised Gabor filters and optimised FIR [48].

Distribution of local descriptors (LBP)

Despite the global structure of textures, accurate discrimination can be achieved by exploiting the joint distributions of pixel values in extremely compact neighbourhoods (e.g. 3×3), outperforming filter banks methods with larger neighbourhoods as reported in [13, 86]. Local descriptors provide a robust texture analysis by describing the pixel intensity values within local neighbourhoods. This section introduces Local Binary Pattern (LBP) [12, 13] descriptors and multiple variants. By computing the occurrence histogram of local descriptors, LBP-like approaches combine structural and statistical methods, largely contributing to an increase of performance in texture analysis. Note that the histogram is referred to as dense orderless pooling. It summarises the occurrence of the local descriptors at every pixel location, regardless of their spatial location, thus discarding the global shapes and image layout. Filter banks and wavelet methods introduced in the previous section as well as descriptors used in vocabulary learning approaches introduced in the next section are also based on the distribution of local descriptors. These approaches are therefore highly related; yet, they are split in this chapter for clarity. The major difference between the filter banks and the LBP-like descriptors include the convolution approach and frequency analysis of spectral methods as well as the compact support (small neighbourhood) of the LBP methods.

The LBP originates from the combination of the analysis of local structures in structural methods and the occurrence analysis of statistical methods. It is also a simplification of the signed differences method previously developed by the same authors in [87]. The original LBP was introduced in [12] with the basic idea to summarise a texture region or image by comparing each pixel with its local neighbourhood (originally 3×3). For each pixel, a binary code is computed by thresholding the neighbouring pixels based on the centre pixel value as shown in Figure 2.8. A histogram of occurrence of binary codes in a texture region or image is then computed resulting in a 256-dimensional feature vector ($2^8 = 256$ possible codes with a 3×3 neighbourhood). This descriptor is very popular due to the simplicity of implementa-

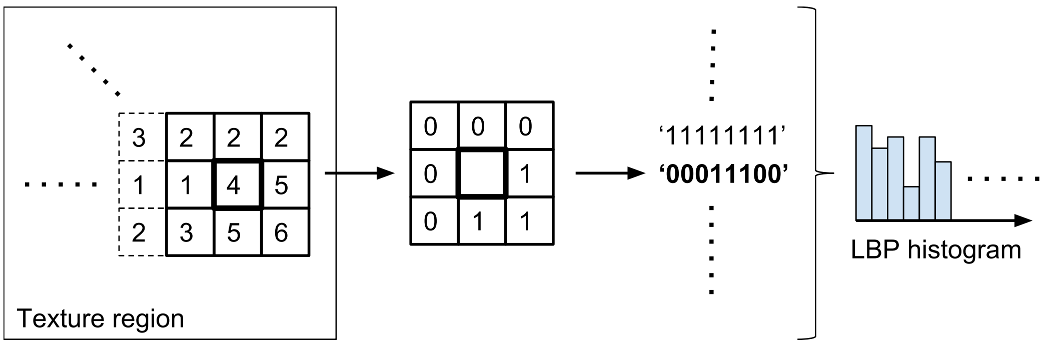


Figure 2.8: The original LBP histogram extraction [12].

tion, its low computational cost and its invariance to monotonic illumination changes. However, the described LBP has several major drawbacks. First, it produces long histograms which are sensitive to image rotation. Secondly, it has a small spatial support (3×3 neighbourhood) which does not extract large-scale textural information. Thirdly, the LBP loses local textural information (e.g. contrast) by considering only the signs of differences of neighbouring pixels. Finally, it is highly sensitive to noise and blurring since slight fluctuations above or below the centre value change the binary code similarly to a major contrast. A large number of variants of the original LBP have been proposed to attempt to overcome these drawbacks. Some of the most popular methods are introduced in the following paragraphs.

A rotation invariant version named LBPROT was achieved by grouping rotated versions of the same binary codes in [88]. This results in a reduced dimensionality of the histogram which represents the occurrences of the 36 unique rotation invariant patterns. However, the quantisation of the angular space with the eight pixels in the circularly asymmetric (squared) neighbourhood is unadapted to the computation of rotation invariance. Moreover, the occurrence of the 36 unique rotation invariant binary patterns varies largely as some patterns are unlikely to occur.

These difficulties were partly overcome in [13], in which several extensions of the LBP descriptor were proposed. The rotation invariant LBP is first improved by using bilinearly interpolated intensity values sampled on a circle of varying radius around the centre pixel. The quantisation of the angular space can be modified by varying the number of interpolated values and local structures can be described at multiple scales by varying the radius. The LBP at a pixel location (x_c, y_c) is computed as follows:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(I_p - I_c)2^p, s(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.12}$$

where $I_c = I(x_c, y_c)$, P is the number of interpolated values considered in the neighbourhood and R is the radius of the circle on which these values lie. A uniform version of the LBP was also introduced in [13] to consider binary patterns that occur most in images and that represent typical local structures such as edges, corners, and spots. In the commonly used $LBP^{riu2}_{P,R}$, superscript ^{ri} refers to the rotation invariant LBP, while ^{u2} considers uniform patterns with at most two transitions from 1 to 0 or vice versa in the circular pattern. All non-uniform patterns are grouped into the same bin. The rotation invariant and uniform LBPs with $P = 8$ are depicted in Figure 2.9, reducing the 256 $LBP_{8,R}$ to 36 $LBP^{ri}_{8,R}$ and 9 $LBP^{riu2}_{8,R}$ patterns. These improvements help obtaining more balanced histogram occurrences and more robust and rotation invariant descriptors as well as reducing the dimensionality of the texture descriptors. The local contrast was used in [13] as a complementarity measure to recover the information lost by considering only the signs of differences of neighbouring pixels.

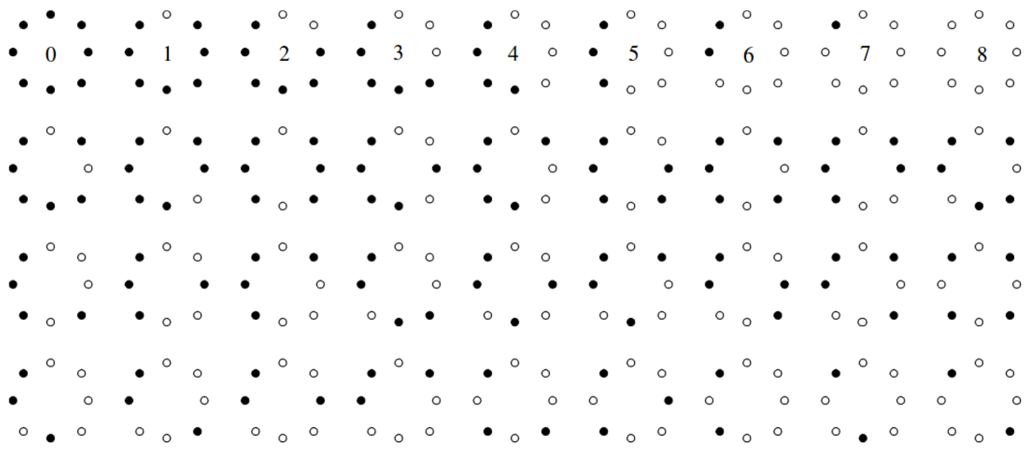


Figure 2.9: The 36 possible rotation invariant $LBP^{ri}_{8,R}$. The nine uniform LBPs ($LBP^{riu2}_{8,R}$) are depicted in the first row. Figure reproduced from [13].

The contrast is measured by computing the histogram of quantised local variances of pixel intensities.

Several variations of the LBP have been developed to improve its discriminative power. The Completed LBP (CLBP) [89] extracts three complementary descriptors. First, the centre pixel provides intensity information using global thresholding. Secondly, a local difference sign-magnitude transform decomposes the local differences into the sign and magnitude components. The intensity, sign and magnitude components are encoded into a CLBP descriptor. Note that the sign component is equivalent to the classic LBP. The Extended LBP (ELBP) introduced in [90] extends the CLBP approach by computing four components. The first two descriptors are based on local intensities (of centre pixels for one and neighbours for the other). The other two descriptors are based on local differences (radial for one and angular for the other). The LBP is extended to Dominant LBP (DLBP) by using the most frequent patterns in the image in order to describe descriptive textural information. This approach considers complex patterns discarded by the uniform LBP in [13] which may be frequent and representative in some textures (e.g. high curvature edges, crossing boundaries or corners). The resulting DLBP descriptor is combined to Gabor-based features to capture complementary global textural information. Note that this approach shares the idea of learning a vocabulary (most occurring patterns) with dictionary learning methods introduced later. A global rotation invariance is obtained in [91] by extracting features from the Fourier transform of the LBP histogram on the entire image.

Many extensions have also been developed to reduce the sensitivity to noise such as the Local Ternary Patterns (LTP) [92]. Neighbouring pixels are thresholded into three values instead of two to consider small intensity variations in a range of values around the centre pixel value $[I_c - k, I_c + k]$, controlled by the constant k. The LTP is computed using two binary patterns to maintain a relatively small dimension of the histogram which is twice as large as the LBP. The Local Phase Quantisation (LPQ) developed in [79] uses local phase information to build texture descriptors insensitive to blur which is frequent in real images generally due to the acquisition (e.g. motion and defocus). The phase information is extracted by a local Fourier transform which is particularly robust to blur. Other descriptors have been extended from the LBP to build robustness to noise and blur including Median Binary Pattern (MBP), Noise Tolerant Local Binary Pattern (NTLBP), Robust Local Binary Pattern (RLBP), Noise Resistant LBP (NRLBP) and Median Robust Extended LBP (MRELBP) [93]. These methods offer various options to encode the local differences with low sensitivity to noise while retaining a high discriminative power of the descriptors.

Second- and higher-order local descriptors named Local Derivative Patterns (LDP) were developed in [94]. They describe the changes of n^{th} order derivative directions in local neighbourhoods.

The Centre-Symmetric LBP (CS-LBP) [95] combines LBP and Scale Invariant Feature Transform (SIFT [17], introduced later) to describe interest regions. Interest point detection is introduced in the next section as it is generally used in texture analysis with vocabulary learning methods. In CS-LBP, pixels are compared to the opposing pixel symmetrically w.r.t. the centre pixel, instead of the centre pixel itself. The robustness to small variations and noise in flat areas is increased by thresholding small differences.

The Weber's Law Descriptor (WLD) [96] is a local descriptor not directly inspired by the LBP. It computes and combines two measures, one based on the differential excitation, the other one on the orientation of local neighbourhoods. The differential excitation is inspired by the human perception of stimuli stated by Weber's law. For each pixel, ratios are calculated between 1) the difference between a neighbour pixel value and the centre pixel and 2) the centre pixel itself. These "Weber's fractions" are summed in local neighbourhoods to detect salient variations in the image. In parallel, the gradient orientation is measured at every pixel location and quantised. A WLD histogram for a given image is then constructed by concatenating sub-histograms which measure the occurrences of differential excitation values for a number of quantised dominant orientations. The sub-histograms are rearranged to group intervals of differential excitations corresponding to a similar variance.

Visual dictionary learning

Visual dictionary or vocabulary learning methods are dataset-dependent as opposed to LBPs or classic filter banks which can be considered as dataset independent predefined dictionaries (except for optimised filters). These approaches are therefore a transition between the hand-crafted shallow feature extraction techniques and deep learning methods which learn and classify abstract concepts. The idea of Bag of Features (BoF) [3, 97], also called bag of visual words, is inspired by the document processing method called Bag of Words (BoW) in which a text document is represented by the occurrence of representative words. Dictionary learning methods in image analysis involve the learning of a visual dictionary. In this context, words are visual patterns represented by local descriptors and a dictionary is made of representative patterns clustered in the feature space.

The early BoF [3] and VZ (Varma and Zisserman) [98] algorithms make the link between filter banks introduced previously and dictionary learning, as the images are filtered by banks of filters. In the VZ algorithm, training images of a given class are first filtered by a filter bank (e.g. MR8). The responses are then clustered with K-means into a visual dictionary. Texture features for a given training or test image are then obtained by 1) convolving it with the same filters as in the training phase and labelling each pixel with the nearest visual word from the dictionary in the filter response space; and 2) building the histogram of occurrence of visual words. The early BoF and VZ algorithms share the K-means clustering, nearest visual word and histogram approaches with many recent vocabulary learning methods.

More recent dictionary learning methods use local descriptors which were not introduced in the previous section as they are not particularly designed for texture analysis apart from their integration with vocabulary learning. These descriptors are generally sparse and involve interest point detection as described below. Therefore, dictionary learning typically involves the detection of keypoints, the extraction of local descriptors, clustering to learn the dictionary (e.g. K-means), and pooling into a global image descriptor (e.g. occurrence count histogram). Several dictionary learning methods are introduced below including BoF [3, 97], FV [7], Improved Fisher Vector (IFV) [11], and Vector of Locally Aggregated Descriptors (VLAD) [20] with details of the major stages involved in their computation.

Note that vocabulary learning methods were not originally developed for texture analysis; yet, they are a suitable extension of classic local descriptors and powerful tools for extracting representative textons as demonstrated in [99]. FV was ported to texture analysis in [100] and BoF, VLAD, IFV, and other vocabulary learning methods in [99]. IFV achieved the best results among the vocabulary learning methods on the tested texture datasets in [99].

Interest point detection

The interest point detection can be seen as the spatial sampling of local neighbourhoods from which descriptors are extracted to describe an image. It can be dense or, more generally in vocabulary learning, sparse, with the aim to extract robust and stable descriptors. Dense sampling does not search keypoints based on the intensity values and patterns but uses a predefined regular grid (e.g. every pixel or pixels separated by a fixed interval). This type of sampling is employed in most of the methods described previously such as LBP and filters which provide a response for every pixel, pooled into a global descriptor. It can also be used with vocabulary learning as in the BoF and VZ algorithm mentioned previously. Note that grid sampling is more adapted to texture images than objects in which sparse fine-grained distinct features may be lost. In contrast, the basic idea of sparse sampling is to find keypoints in the image which exhibit particularly well localised patterns (e.g. corners or blobs). Such detection was primarily developed to match interest points in multiple images for tracking, stereo matching, and calibration. It can be used in texture analysis to localise keypoints where robust texture descriptors (i.e. repeated patterns or textons) can be extracted.

The Harris detector is a popular corner detection method based on the eigenvalue analysis of the structure tensor (second-moment matrix) which was used in a BoF method in [101]. Corners are relatively robust and well localised keypoints as a translation in any direction results in a large change of pixel values, unlike constant areas and edges which suffer from the aperture problem. The Harris detector originates from a weighted Sum of Squared Differences (SSD, similar to autocorrelation) in which the shifted image intensities are approximated by a Taylor expansion, i.e. based on the partial derivatives I_x and I_y at a single point. To that end, an eigenanalysis of the structure tensor is performed at each pixel location. A standard Harris detector is not scale invariant, although it is possible to perform it in a multi-scale analysis. More complex methods are therefore introduced in the following text to obtain scale invariance required in keypoints matching and recognition as well as in many texture analysis tasks.

Alternatively to the Harris detector, robust scale invariant keypoints can be detected by a scale space analysis using Difference of Gaussians (DoG) [17]. A DoG is performed by subtracting a smoothed image from another less smoothed image, resulting in a scale band-pass.

$$D(x,y,\sigma) = I(x,y) * G(x,y,k_i\sigma) - I(x,y) * G(x,y,k_j\sigma) \tag{2.13}$$

where $G(x,y,k_i\sigma)$ and $G(x,y,k_j\sigma)$ are Gaussians of standard deviation $k_i\sigma$ and $k_j\sigma$. A scale space $(x,y,k\sigma)$ is obtained by stacking DoG responses at multiple scales of the full-size image as well as successively downsampled images. Extrema are sought in local neighbourhoods in the derived scale space as they exhibit robust scale invariant keypoints. A Taylor series approximation of $D(x,y,\sigma)$ and an eigenvalues analysis are then used to remove ambiguous points including low contrast and edge points (due to the aperture problem).

Another approach to scale space analysis is based on an approximation of the Hessian matrix and used to speed up the detection of blob keypoints in [102]. A Hessian matrix involves the convolution of the image with the second-order derivatives of Gaussian filters w.r.t. x and y as follows:

$$\begin{pmatrix} I(x,y) * \frac{\partial^2 G(x,y,\sigma)}{\partial x^2} & I(x,y) * \frac{\partial^2 G(x,y,\sigma)}{\partial x \partial y} \\ I(x,y) * \frac{\partial^2 G(x,y,\sigma)}{\partial x \partial y} & I(x,y) * \frac{\partial^2 G(x,y,\sigma)}{\partial y^2} \end{pmatrix} \tag{2.14}$$

These second-order derivatives are approximated by simple box filters and a scale space analysis is performed by applying a set of filters of varying scales. Keypoints are then detected by non-maximum suppression of the determinants of Hessian matrices in $3 \times 3 \times 3$ neighbourhoods in the derived scale space.

Other robust and fast interest point detection methods found in texture analysis include LoG and Features from Accelerated Segment Test (FAST) [103], a high-speed corner detection algorithm used in [104].

Local descriptors

Classic local texture descriptors such as LBP and filter banks were introduced in the previous sections. Other types of local descriptors, not originally designed for texture analysis, are generally used in vocabulary learning such as SIFT and SURF as explained in the following paragraphs. Local descriptors are computed for each interest point.

The Histogram of Oriented Gradients (HOG) [105] derives local histograms of gradient orientations. The image is divided into overlapping blocks and cells of pixels in which the gradient orientation is computed and quantised. Orientations and magnitudes are computed from the gradient image obtained with simple x and y derivative filters ($[-1,0,1]$ and $[-1,0,1]^T$ respectively).

SIFT local descriptors [17], commonly used in vocabulary learning methods (e.g. in BoF, FV, IFV and VLAD), are based on local gradient orientation. The keypoints are commonly detected with DoG as explained previously, although Harris or other keypoint detectors can be used [101]. First, the keypoint dominant orientation is computed using a weighted histogram of quantised gradient orientations in local neighbourhoods. A vector descriptor is then formed by computing weighted histograms of relative orientations w.r.t. the dominant orientation in blocks of pixels (e.g. 4×4) within neighbourhoods (e.g. 16×16). The dimension of the derived descriptors is 128 (16 histograms of 8 orientation bins).

SURF local descriptors are inspired by SIFT descriptors and also describe relative orientations of local neighbourhoods. The keypoint detection is performed in the scale space based on the Hessian matrix explained previously, and the orientation assignment and encoding is based on Haar wavelet responses. Note that these descriptors were primarily developed for matching images and therefore provide important invariances or robustness to geometric and photometric transformations relevant for many texture problems.

Other descriptors used in vocabulary learning, which are or can be ported to texture analysis, include PCA-SIFT [106], Binary Robust Independent Elementary Features (BRIEF) [107] used in [104] and the rotation invariant descriptors called spin image and RIFT (Rotation-Invariant Feature Transform) developed in [108].

Descriptors clustering

This step is the major difference to most methods described in the previous sections which compute dense local descriptors (e.g. LBP, response to filter banks, etc.) and directly pool these descriptors into texture features based on their distribution, energy response, etc. In vocabulary learning, representative patterns are learned from the training set by clustering the local descriptors into K clusters in the feature space. Local descriptors are typically clustered in an unsupervised manner using K-means or Gaussian Mixture Models (GMMs). In the BoF [97, 101] and VLAD approach [20], K-means is used to cluster the descriptors. Each cluster centre is then considered as a “word” in a visual dictionary (i.e. representative of multiple local patches). A repeated pattern or texton is thus described by descriptors clustered into the same visual word in the feature space.

The FV [7] and the IFV [11]) can be seen as an extension of the BoF method [97, 101] using soft clustering. The clustering in these approaches can be computed by fitting a GMM to the distribution of descriptors instead of K-means clustering which cannot capture overlapping distributions in the feature space as it considers only distances to cluster centres. A GMM considers both cluster centres and covariances which describe the spread. The Gaussian distributions are optimised with Expectation Maximisation (EM) similar to the K-means algorithm, while data points are assigned to clusters with soft probabilities. The GMM therefore generates a probabilistic visual dictionary, as opposed to the hardcoded dictionary obtained with K-means.

Pooling and normalisation methods

While vocabulary learning methods are often used for matching images and regions by comparing individual words or pairs of global image descriptors, it can also be used to build a global descriptor of an image from the learned dictionary, for instance, for texture classification. The pooling of local descriptors into a global descriptor is similar to the derivation of global descriptors from LBPs and filter responses. A typical pooling method is the histogram of occurrences of the visual words from the learned dictionary [3]. The occurrence is often based on the nearest cluster centre (with K-means clustering) in the feature space of each local descriptor of an image. It results in a feature vector which summarises the occurrence of visual patterns.

VLAD encoding considers distances of local descriptors to the representative visual words (cluster centres) instead of the simple occurrence used in BoF. For each visual word, the distance to the cluster centre is accumulated in each dimension for all the local descriptors in the image which are closest to this visual word.

As mentioned previously, FV and IFV cluster local descriptors using GMM. A FV is then computed by assigning each local descriptor in an image to a visual word in the learned dictionary, and by computing the gradients of the soft assignments w.r.t. the mixture weights, means, and covariance matrices. In this way, the FV does not only encode the occurrence of local descriptors like a BoF (probabilistic occurrence related to the gradients w.r.t. the mixture weights) but also higher-order statistics (gradients w.r.t the means and covariance matrices). The FV thus contains significantly more information than the BoF and VLAD vectors and its dimension is significantly larger. For comparison, with K visual words of dimension D (e.g. $D = 128$ for SIFT), the BoF vector dimension is K, VLAD's is DK and FV's is $2DK$.

Normalisation of the obtained vectors is often computed to improve the classification or matching results. The vectors are generally normalised by the number of local descriptors in the image to avoid the dependence on the sample size. Two other normalisation methods were developed in the IFV [11], namely ℓ_2 normalisation and power normalisation. Despite the multi-scale analysis in the extraction of local descriptors, FV still depends on the proportion of image-specific information (i.e. background information) and consequently depends on the scale of the observed scene. To remove this dependence, the vector is ℓ_2 normalised. The power normalisation refers to the signed square root applied to each dimension of the image descriptor vectors (i.e. each element x_i is transformed into $sign(x_i)\sqrt{x_i}$) to reduce the sparsity of the representation. This is motivated by the poor measure of similarity of sparse vectors with the dot-product used for classification (e.g. with SVM in [11], see Section 2.2.3).

The described pooling methods are suited to texture analysis as they are orderless, i.e. they do not carry layout or global shape information since the spatial position of local descriptors is discarded, for instance by the histogram of occurrence. BoF, SIFT and VLAD were ported to texture classification in [99], in which a significant improvement was reported from classic hand-crafted methods.

Vocabulary learning methods have partly motivated the use of CNNs in texture analysis, sharing the idea of extraction and pooling of learned descriptors. Note that FV encoding has also been applied to the output of CNNs, as described in Section 2.2.4.

2.2.3 Texture analysis problems

The described texture features can be extracted for every pixel or every interest point and pooled into a global descriptor for a texture region or an entire image depending on the task. Note that feature selection [12, 87] and dimensionality reduction [78, 106, 8] (e.g. PCA) can be used to avoid redundancy, a curse of dimensionality and overfitting the training data. This section introduces four major texture analysis problems, namely texture classification, segmentation, synthesis and shape from texture.

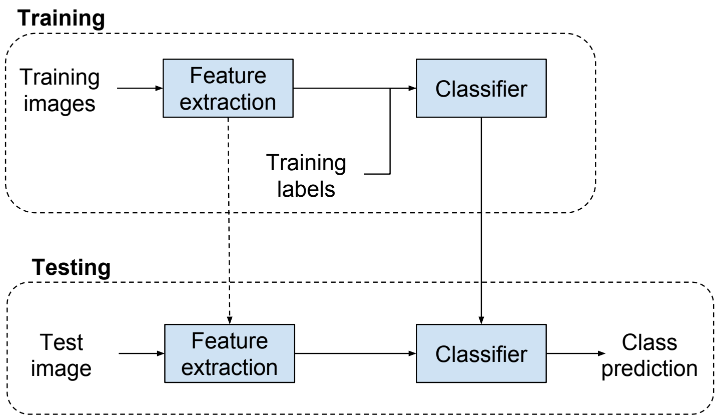


Figure 2.10: Training and testing phases in a texture classification framework. Note that the feature extraction is learned from the training data in optimised filters and dictionary learning methods as represented by the dashed arrow.

Texture classification

In texture classification, a set of training images is used to train a model in a supervised manner as shown in Figure 2.10. A trained model is then used to classify unknown texture images. The feature extraction methods introduced in the previous section build a N-dimensional feature vector (e.g. statistical/structural measures or histogram) for a given image. Texture classification methods extract these texture features for all the training samples. Samples from the same class are ideally clustered in the feature space. A decision rule (classifier) is then learned to label an unknown test image to a given class based on its projection into the feature space. Several distance measures and classifiers have been used in texture classification. Distance measures estimate the similarity of descriptors in the feature space (e.g. Euclidean, Mahalanobis, Manhattan and chi-square distances). A distance measure is typically used in a K-NN classifier which assigns a test image to the majority voting of the K nearest training samples in the feature space.

A linear classifier makes a decision rule for a given image based on a linear combination of its feature vector. A linear SVM is a binary classifier which constructs a hyperplane in the feature space to separate two classes represented by multiple training samples. The hyperplane maximises the margin (calculated as vectors) on both sides to the nearest samples. While linear classifiers have the advantage of low complexity, the data is often not linearly separable in which case a non-linear SVM is more appropriate. The non-linear SVM maps the feature space into a higher-dimensional space in which the data is linearly separable. Several kernel functions are used to map the feature space including polynomial and radial basis function. Texture classification problems generally involve more than two classes. A multi-class SVM is generally constructed by considering multiple binary classification problems including one-against-all, one-against-one, and Directed Acyclic Graph SVM (DAGSVM) methods [109]. Alternatively, SVMs can be trained with gradient descent (introduced in Appendix A.2.3) [110]. Neural networks such as MLP introduced in Appendix A can also be trained for classification with backpropagation using feature vectors as inputs. Other classification methods used in texture classification include naive Bayes [101, 111] and Adaptive Boosting (AdaBoost) [38].

Most texture classification datasets evaluate the recognition of materials, including kth-tips-2b [22, 112], Kylberg [31], CUReT [113], and UIUC [108]. These datasets contain texture images with ground truth class labels. It is common practice to split a dataset into training (with ground truth) and testing (unknown) sets to evaluate the performance of a classification algorithm. Note that on very large image classification datasets like ImageNet [50], training, validation and test sets are predefined. Several methods exist to evaluate and compare the performance of classification algorithms, while avoiding to overfit a single test set. The dataset can be randomly split into training and testing sets (e.g. 80% and 20% respectively) and repeated multiple times to average the accuracy and report the standard deviation on the test set. This approach, also referred to as Monte Carlo cross-validation, does not ensure that each sample is used for testing. Other cross-validation methods ensure that each sample is used once for testing, providing a powerful measurement of performance. K-fold cross-validation is a non-exhaustive method in which the dataset is partitioned into K folds of equal sizes (typically $K = 10$). Each fold is used once for testing and the remaining folds for training. Leave-N-out cross-validation is an exhaustive cross-validation method, commonly used with $N = 1$ (leave-one-out) in which each sample is used once for testing and the rest for training. Leave-one-out is thus a particular case of K-fold where K is equal to the number of samples.

The performance of a texture classification algorithm is typically measured by the average accuracy and standard deviation over the various splits described above. A confusion matrix can also provide meaningful information about the performance. The confusion matrix of an N-class problem is an $N \times N$ matrix which represents the true and false classification of each class. Other performance measures can be used for datasets with unbalanced numbers of samples per class.

Texture segmentation

Texture segmentation aims at partitioning an image into regions of homogeneous texture properties. It therefore requires the classification or clustering of every pixel in the image. It can be supervised (providing training samples of the textures to segment) [26, 114, 115] or unsupervised [21, 26, 57, 65, 68, 116, 117]. In both cases, texture descriptors introduced in Section 2.2.2 are typically obtained for every pixel or superpixel and either classified (supervised) or clustered (unsupervised) in the feature space. Local spectral histograms from Gabor and other filters responses are commonly used features in texture segmentation [116–118]. Model-based segmentation such as MRF and GMRF are also well suited for this task [119, 120]. The segmentation of texture descriptors can be performed, among others by curve evolution with level-set optimisation [114], region growing and merging [121] and functional minimisation (Mumford-Shah functional) [116, 117]. Other basic methods to cluster texture descriptors in the feature space include K-means, mean-shift, GMM, region splitting and watershed [38].

Texture segmentation benchmarks can be mosaics, i.e. artificially created from segments of multiple texture images, or real images with multiple texture regions. The Brodatz texture dataset is commonly used to create mosaics in the literature [21, 65, 68, 78]. Note that using mosaics automatically provides precise ground truth. The Prague texture segmentation benchmark [26] enables the testing and comparison of algorithms on a range of supervised and unsupervised segmentation tasks with various texture mosaics.

Commonly used performance metrics of segmentation algorithms can be grouped into pixel-wise, region-based, consistency and clustering measures. Pixel-wise measures are based on counts of wrongly interpreted pixels (e.g. classified as class i but different ground truth class) and wrongly assigned pixels (e.g. ground truth i but classified as another class). These metrics include the Omission (O) error (a ratio of wrongly interpreted pixels), Commission (C) error (a ratio of wrongly assigned pixels), weighted average Class Accuracy (CA), recall (CO, the average correct assignment) and precision (CC, overall accuracy). Note that the ground truth classes of pixels are compared to segmented results using the Munkres algorithm in the unsupervised case. Region-based measures compare segmented and ground truth regions $R_i, i = 1,...,M$ and $\bar{R_j}, j = 1,...,N$ respectively, where M and N are the number of segmented and of ground truth regions. These metrics include Correct-, Over-, and Under Segmentation (CS, OS and US) as well as Missed-, and Noise Error (ME and NE). A region R_m is considered CS if and only if $R_m \cap \bar{R_n} \geq k\bar{R_n}$, where k

is a threshold parameter (e.g. 0.75). OS is a count of regions $\bar{R_n}$ split into smaller regions R_m (and vice versa for US). ME and NE are counts of regions $\bar{R_n}$ and R_m respectively that do not belong to CS, OS and US. Other metrics include Global- and Local Consistency Error (GCE and LCE) and clustering measures (Mirkin metric, Van Dongen metric and variation of information). A detailed description of these performance measures is provided in [26].

Texture synthesis

Texture synthesis involves the generation of a texture image or region from a texture sample. It is commonly used for image inpainting, computer graphics, and image compression. Model-based methods are well suited for image synthesis by building a parametric model which captures the statistical properties of a texture image and allows it to generate visually similar images with identical properties. Typical models used for texture synthesis include MRF [27] and fractals [67] introduced in Section 2.2.2. Filter banks and wavelets have also been used for texture synthesis, modelling images by statistics on responses and wavelet coefficients [28]. Finally, recent deep learning texture synthesis methods are introduced in the Section 2.2.4.

Shape from texture

Shape from texture, originating from [29], is used to reconstruct the shape of a 3D object from a 2D image. The distortion of a generally regular surface texture is analysed to infer the orientation and shape of a surface. Effects of the visual geometry on the 2D texture appearance include foreshortening, compression, scaling and changes in area and density. Common analysis methods include measures of gradients of texture appearances in the 2D plane and model-based (e.g. isotropy texture model) methods [30].

2.2.4 Deep descriptor and deep learning in texture analysis

This section describes recent works on CNNs and deep learning in texture analysis. Neural networks, deep learning and CNNs are introduced in Appendix A to which the reader may refer if not familiar with the concepts used here.

In [122], the authors argue that the dimensionality of texture datasets is too large to classify them with deep neural networks without explicitly extracting hand-crafted features beforehand, as opposed to digits or object datasets which lie on a lower-dimensional lattice. They affirm that neural networks must therefore be redesigned to learn texture features similar to GLCM and Haralick features. Recent work on CNNs applied to texture analysis, however, illustrates that they cope very well with texture datasets [8, 25, 36, 123]. It was shown that convolutional architectures are, subject to minor modifications, well designed for the analysis of texture and that they largely improve the state of the art in this field.

Basic CNN architectures have been applied to texture recognition such as [124], in which a simple four layer network was used in the early stages of deep learning to classify simple texture images. More recent CNNs were applied to forest species images with high texture content [125]. Transfer learning between texture datasets was studied in [126] with classic CNN architectures and applied to the forest species classification, similar to a texture recognition problem. While more complex and more accurate than [124], these approaches still do not take the characteristics of texture images (i.e. statistical properties, repeated patterns and irrelevance of the overall shape analysis) into consideration as they are simple applications of a standard CNN architecture to a texture dataset. A Recurrent Neural Network (RNN) approach (see Appendix A.2.5) was used for texture classification in [127] and texture segmentation in [115]. This approach, however, does not benefit from the weight sharing and local connectivity of CNNs to efficiently detect texture patterns and requires data augmentation to learn simple invariances. A wavelet scattering convolution network (ScatNet) was developed in [128] using hand-crafted wavelets as convolution filters and extended to affine invariance in [129]. The PCA Network (PCANet) [130] inspired by ScatNet uses a cascade of PCA to learn filter banks in a CNN-like architecture with binary quantisation used for non-linearity. Block-wise histograms of the quantised responses are pooled and classified using SVM or softmax.

Following these early applications of CNN to texture images, several methods have been developed with a general idea of discarding the overall shape analysis of CNNs by an orderless pooling of feature maps with excellent results as described in the following paragraphs.

Deep Convolutional-network Activation Features (DeCAF) [131], which extract the penultimate fully-connected layer of AlexNet for SVM classification, were ported from object recognition to texture classification in [99]. Texture descriptors were densely extracted from a convolutional network with the Fisher Vector CNN (FV-CNN) in [8]. Inspired by DeCAF, a CNN (VGG-M or VGG-VD) pre-trained on ImageNet [50] is used as a feature extractor. The output of the last convolution layer is used in a FV encoding classified with one-vs-rest SVM. The overall shape information is discarded in this analysis by replacing the fully-connected layers by the FV orderless pooling and SVM classification. As the network is only being used for feature extraction, the convolutional network is not finetuned and does not learn from the texture dataset. However, due to the domain transferability of filters pre-trained on ImageNet, they achieve impressive results on both texture recognition and texture recognition in clutter datasets. The FV-CNN is also combined with shape analysis by extracting the outputs of the penultimate fully-connected layer (called FC-CNN, similar to DeCAF), demonstrating the complementarity of the shape and texture analysis. Finally, the FV-CNN can efficiently be used in a region-based approach as it requires computing the convolution output once and pooling the desired regions with FV encoding. In [123], the FV-CNN is improved with a dimensionality reduction to reduce the redundancy and increase the discriminative power of the features extracted from the CNN prior to SVM classification. This approach involves extracting the output of a pre-trained CNN, FV encoding, training an ensemble of fully connected neural networks for dimensionality reduction and training an SVM for classification.

A Texture CNN (T-CNN) was developed in [25] which includes an energy layer to extract the dense response to intermediate features in the network, improving the results on texture classification tasks while reducing the complexity compared to classic CNNs. The complementarity of texture and shape analysis is shown with an end-to-end CNN training scheme. A framework splitting the images and using the T-CNN with a voting score approach was developed in [35, 51] for the classification of biomedical texture images. A fully convolutional approach was developed for the segmentation of texture regions in [53]. These three methods [25, 51, 53] will be presented in the thesis as part of the main contributions.

A bilinear CNN model was developed in [132] for fine-grained recognition (i.e. visually and semantically very similar classes), combining two CNN streams to extract and classify local pairwise features in a neural network framework. One stream made of convolution and pooling layers works as an object recognition network (i.e. the “what”), while the other stream analyses the spatial location of the object in the image (i.e. the “where”). Their output feature maps are multiplied using outer product and densely pooled across the image by summing the extracted features. Robust image descriptors are obtained which capture translational invariant local feature interactions. The developed architecture can also generalise several classic orderless pooling descriptors including VLAD and FV in a deep learning framework. This method is successfully applied to texture classification in [36] and obtains slightly better results than the FV-CNN while being trained end-to-end.

A Deep Texture Encoding Network (Deep-TEN) was introduced in [133] by integrating an encoding layer on top of convolution layers, also generalising orderless pooling methods such as VLAD and FV in a CNN architecture trained end-to-end.

Rotation invariance was embedded in a shallow CNN in [134] by tying the weights of multiple rotated versions of filters for texture classification. While rotation invariance of simple texture descriptors can be learned and pooled orderless with average pooling, the benefit over hand-crafted rotation invariant descriptors is limited by the use of a single layer.

In [93], multiple deep texture descriptors were evaluated including FV-CNN, ScatNet and PCANet, and compared to several variants of LBP descriptors. As expected, the deep convolutional descriptors obtain the best results, at the cost of a much higher computational complexity as compared to LBP variants.

Finally, deep networks have also been applied to texture synthesis. A pre-trained CNN was used in [135] to compute statistics (correlation between feature maps) at multiple layers using a source image as input. A new texture image is then generated by optimising an initial random image via gradient descent (see Appendix A.2.3) to obtain similar statistics at multiple depths. A spatial GAN (see Appendix A.3.6) was used in [37] to synthesise texture images of arbitrary sizes by replacing the random noise vector generally used as input to the generator by a spatial tensor.

2.3 Dynamic texture analysis

DT is an extension of texture in the temporal domain. Various spatial texture descriptors introduced in the previous section have therefore been extended to the spatiotemporal domain to capture temporal variations such as motion and deformation. The major DT analysis problems include recognition [14, 45–47, 136–147], segmentation [40], synthesis and compression [45]. DT recognition datasets commonly used in the literature include UCLA [45], DynTex [24] and Dyntex++ [39]. More details on these datasets are provided in Section 4.3.1.

2.3.1 Classic dynamic texture analysis

Difficulties in DT analysis arise from the large range of phenomena resulting from natural scenes and recording methods including scale, illumination and rotation variation as well as static and moving cameras. Most classic DT analysis approaches that attempt to overcome all or some of these difficulties can be classified into five categories, namely statistical, local descriptors, model-based, spatiotemporal filtering and motion-based.

Statistical and local descriptor approaches extend standard spatial texture methods such as GLCM and LBP to the spatiotemporal analysis of DTs. The LBP was extended to DTs in [148], considering a spatiotemporal neighbourhood. This method was improved and its computational cost reduced in [14] by extracting histograms on Three Orthogonal Planes (LBP-TOP): *xy* which is the classic spatial texture LBP as well as *xt* and *yt* which consider temporal variations of the pixel intensities. Several variants of the LBP-TOP have also been proposed including the extension of LPQ to Three Orthogonal Planes (LPQ-TOP) [138]. The LBP-TOP was combined to a Weber Local Descriptor on Three Orthogonal Planes (WLD-TOP) in [40] to analyse local spatiotemporal neighbourhood and combine it to motion features for DT segmentation. More recently, a noise resistance feature was incorporated to the LBP-TOP with a Weber's law threshold in the creation of the patterns [149], achieving high accuracy on the Dyntex++ and UCLA datasets. The main drawbacks of the LBP-TOP approaches include the limited amount of information that it is able to extract with the hand-crafted patterns and the poor robustness to camera motion. The latter is addressed by computing the discrete Fourier transform of LBP-TOP histograms in [150] resulting in a rotation invariant descriptor robust to changes in viewpoint.

Motion-based methods exploit the statistical properties of the extracted motion between consecutive frames of the DT sequences. Several motion extraction methods have been used for the analysis of DT including complete flow [40] and more commonly normal flow [46, 140, 141]. Statistical features are extracted from the motion to describe its spatial distribution such as GLCM [151, 46], Fourier spectrum [46], difference statistics [46], histograms [40] and other statistics calculated on the motion field itself [140] and on its derivatives [141].

Model based methods aim to estimate the parameters of a Linear Dynamical System (LDS) using a system identification theory in order to capture the spatial appearance and dynamics of a scene. Originally designed for synthesis, the estimated parameters can be used for a classification task [45]. Positive results were obtained in the learning and synthesis of temporal stationary sequences such as waves and clouds in [45]. The model-based approach, however, raises several difficulties such as the distance between models lying in a non-linear space of LDSs with a complicated manifold. It also offers a poor invariance to rotation, scale, and illumination. In [143] impulse responses of state variables learned during the system identification were used to capture the fundamental dynamic properties of DTs. This approach was developed to deal with the problems of non-linear space, segmentation and presence of multiple DTs in a sequence. The view-invariance problem was tackled in [142] using a Bag of dynamical Systems (BoS) similar to a BoF with LDSs as feature descriptors. In [136], a new distance between LDSs was defined in the non-linear space based on the notion of aligning LDSs. It enables the computation of the mean of a set of LDSs for an efficient classification, particularly adapted to large-scale problems.

Filtering and transform approaches for texture analysis were also extended to the spatiotemporal domain for the analysis of DTs [47, 144–147]. Spatiotemporal oriented filters (3D Gaussian third derivatives) were implemented in [47] to extract oriented energy features which describe intrinsic properties of the DTs. A spatiotemporal directional number transitional graph was proposed in [152]. Graph-based descriptors are computed from the response to a set of spatiotemporal oriented local filters (2D and 3D Kirsch compass masks) to represent the signature of DTs. In [144], a 3D DT-CWT used the spatial orientation and motion selectiveness of wavelets to combine spatial and dynamic analyses. Other filtering methods have been developed including Wavelet Domain Multifractal Analysis (WDMA) [146], Spatiotemporal Oriented Energy (SOE) [147], and spatiotemporal Gabor filters [145].

Finally, many methods combine two or more approaches for their complementarity in the spatiotemporal analysis [39, 40, 139, 140]. In [39], the LBP was combined with a pyramid of HOG and an LDS model approach to jointly analyse the spatial texture, the spatial layout and the dynamics of DT sequences. More recently, an ensemble SVM was used in [139] to combine static features such as LBP and responses to Gabor filter banks with temporal model-based features.

2.3.2 Deep learning in dynamic texture analysis

Similarly to the analysis of static textures, several attempts have been made to apply deep features and deep learning methods to videos and/or DT analysis.

End-to-end convolutional networks with 3D filters were developed in [153], capturing spatial and temporal information for human action recognition. In [154], the 3D convolution and pooling were generalised to cope with large video datasets. Excellent results were obtained on action, scene and object recognition. In [155], several CNN architectures were evaluated on a video classification task including classic single-frame CNN, two-stream single frame networks with various fusion approaches and 3D convolution filters. However, their attempt to combine motion analysis to the classic spatial convolutional network by using stacks of images as input resulted in a surprisingly modest improvement. Although a few sequences benefit from the motion analysis, the latter increases the sensitivity to camera motion.

A two-stream CNN method was developed in [156] including spatial and temporal analysis for action recognition. In this architecture, a spatial network is trained to recognise still video frames while a motion network is trained on dense optical flow. The two networks are trained separately and their softmax scores are combined by late fusion. This work demonstrated the complementarity of spatial and temporal analysis with deep learning in video classification. A long-term recurrent CNN was introduced in [157] to learn compositional representations in space and time for video analysis. This recurrent approach is able to learn complex temporal information from sequential data on account of a temporal recursion. While this method is appropriate for activity recognition, image captioning, and video description, it is not designed for DT analysis in which one is more interested in statistics of the distribution of pixel intensities (in time and space) than in the sequential detection of particular events.

The semantic selectiveness of spatial convolution filters was used in [158] to discard the background in crowd scenes followed by a combination of spatial and temporal convolutions in a CNN architecture for crowd video understanding. In [159], the same authors used a FCN (see Appendix A.4.5) on spatial and temporal slices and on the motion extracted from the sequences, with various fusion strategies in order to create a per-pixel collectiveness map from crowd videos. These approaches share some similarities with the method proposed in the thesis; however, they were designed for the analysis of crowded scenes with assumptions (e.g. presence of background) and methods (e.g. FCN for collectiveness map) which do not apply to the DT recognition tasks considered in Chapter 4.

While the deep learning methods for video analysis introduced so far were mainly developed for human action recognition, several recent neural network approaches have focused on classifying DTs. Temporal gradients and Group Of Pictures (GOP) were used in [160] as inputs to CNNs to classify DTs. In order to use the pre-trained AlexNet, the authors dilate and warp the 50×50 images to the AlexNet input size 227×227. This pre-processing step results in a waste of computation and a network architecture which is not adapted to the input images. Similar to the spatial approach in [8], in which CNNs are used to extract features from static texture images, a CNN extracts features from the DT database in [41]. A pre-trained CNN is used as a feature extractor to obtain mid-level features from each frame of the sequence and then compute and classify the first and second-order statistics. A variant of stacked auto-encoder was trained in [161] to learn video dynamics which can be more complex than those based on LDS estimation. The deep encoded dynamics can be used similarly to classic model-based methods to synthesise and recognise DTs and segment motions. The PCANet was extended to DT in [162] using filters learned via PCA on three orthogonal planes. DT features are pooled via histogram count and classified with linear SVM. A deep dual descriptor was developed in [163] which uses a pre-trained CNN (VGG-VD) to first extract key frames from the sequence. A visual dictionary of static features is then learned from key frames, while a dictionary of dynamic features is learned from segments of adjacent frames in the temporal neighbourhood of the key frames.

Finally, the T-CNN [25] was ported to DT classification in [52] by training networks end-to-end on three orthogonal planes and using a late fusion approach at test time as described in Chapter 4.

Chapter 3

Convolutional networks for texture classification

3.1 Introduction

This chapter is dedicated to the classification of texture images with deep learning. A new CNN architecture is introduced, which is trained end-to-end to learn texture descriptors and their classification. Intermediate convolution layers are pooled similarly to an energy response within the CNN, allowing forward and backward propagation to learn and classify robust texture features. The learned features are visualised and analysed together with their domain transferability. Networks from scratch as well as pre-trained are evaluated on both texture and object datasets.

The experiments demonstrate that simple networks with reduced number of neurons and weights are able to obtain competitive results on texture recognition datasets. One of the major trends in the community of deep neural networks is to develop more and more complex networks, using the increasing power and memory of computers and GPUs to train very deep and computationally expensive networks. The interest and use of CNNs, however, is not limited to powerful desktop computers and designing efficient networks while restraining their size is important for mobile and embedded computing as mentioned in [164]. In consequence, the work presented in this chapter is not focused on competing with the state of the art in texture recognition but rather on designing simple architectures specifically designed for textures as well as gaining insight on how CNNs learn texture feature.

To summarise, the main contributions of this chapter can be described as follows:
(1) A simple CNN is developed with reduced complexity to extract, learn and classify texture features;
(2) the domain transferability of pretrained features is evaluated and compared with randomly initialised parameters;
(3) various network depths are tested when applied to texture and object classification

datasets;
(4) texture and shape analyses are combined within a new network architecture;
(5) an application of the developed texture specific CNN is proposed on the classification of biomedical tissue images in a framework adapted to large images.

The rest of this chapter is organised as follows: The architecture of the proposed texture network and its combination with a classic CNN is introduced in Section 3.2. Section 3.3 describes the texture classification datasets and experimental protocols. The results on commonly used texture datasets are detailed in Section 3.4. Finally, an application to tissue images is presented in Section 3.5.

3.2 Material and Methods

This section describes the developed architecture named Texture CNN (T-CNN), and its combination with a classic CNN approach.

3.2.1 Texture CNN

Convolutional networks naturally densely extract features by the weight sharing and local connectivity of the convolution layers (see Appendix A.4.3). These layers can be compared to filter bank methods widely used in texture analysis (see Section 2.2.2). While these filters are pre-designed in classic filter bank methods, the power of CNNs is to learn meaningful features through gradient descent. A simple network architecture is developed based on this observation. As explained in [8], the global spatial information is of minor importance in texture analysis as opposed to the necessity of analysing the global shape for an object recognition task[1]. Therefore, dense texture descriptors are pooled from the output of a convolution layer. The proposed network is derived from AlexNet [15], while the same approach is briefly evaluated with the deeper GoogleNet architecture [164] in Section 3.4.7. A new energy layer is introduced in the following description, and implemented in multiple configurations with varying numbers of convolution layers. Each feature map of the last convolution layer is simply pooled by calculating the average of its activated output. The forward and backward propagation of the energy layer are similar to an average pooling layer over the entire feature map, i.e. with a kernel size equal to the size of the feature map.
The forward computation of the energy layer is computed as:

$$e_j = \frac{1}{WH} \sum_{x_1=1}^{W} \sum_{x_2=1}^{H} f_j(x_1, x_2) \tag{3.1}$$

[1]Note that in this chapter, "shape" refers to the global shape of objects rather than low-level shapes which are part of an object.

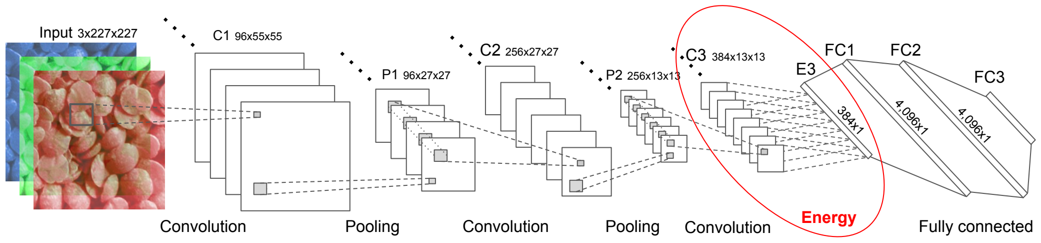


Figure 3.1: A T-CNN architecture with three convolution layers (T-CNN3).

where $\boldsymbol{e} \in \mathbb{R}^N$ is the vector output, $f_j(x_1,x_2)$ is the j^{th} input feature map at spatial location (x_1,x_2). Lowerscript j enumerates the N input feature maps and W and H are the number of columns and rows of the input feature maps respectively. This results in one single response per feature map, similar to an energy response to a filter bank, whereas learned features of varying complexity are used instead of fixed filters. Note that it is referred to as *energy* in reference to the filter bank approach even though an averaging of the responses is performed. The gradients are backpropagated through the energy layer by computing:

$$\frac{\partial e_j}{\partial f_j(x_1,x_2)} = \frac{1}{WH} \tag{3.2}$$

The architecture of the T-CNN3 includes three convolution layers C1, C2 and C3 as illustrated in Figure 3.1. The vector output of the energy layer (E3) is simply connected to a fully-connected layer FC1, followed by two other fully-connected layers FC2 and FC3. Similar to other networks, FC3 is of size equal to the number of classes and the probabilities of the classes are calculated with a softmax layer. Linear rectification, normalisation and dropout are used in the same way as in AlexNet. Five architectures are experimented, T-CNN1 to T-CNN5, with one to five convolution layers respectively. The energy is pooled from the output of the last convolution layer. The support of the filter responses pooled by the energy layer is equivalent to the receptive field which increases throughout the network, from 11×11 in C1 to 163×163 in C5. However, the "effective" receptive field may be considered smaller since pixels in the centre of the receptive field have more influence in a neuron's activation than those on the side.

The complexity of the proposed T-CNN approach is largely reduced when compared to a classic AlexNet or VGG [18] network. Convolution layers are removed from the T-CNN architectures (except T-CNN5 which has the same number of convolution layers as the original AlexNet) and the number of parameters between the energy layer and the fully-connected layer FC1 is largely reduced as compared to the classic full connection of the last convolution layer. The number of trainable parameters of the different networks used in this chapter are indicated for comparison in Table 3.1. These numbers account for the weights and biases of the networks applied to ImageNet (1,000 classes).

3.2.2 Details of the network

The networks are implemented with Caffe[2] [165] and derived from AlexNet. The number of feature maps, kernel sizes, etc. are kept unchanged from AlexNet for comparison. However, it is possible to reduce the size of the fully-connected layers of the T-CNN by a factor greater than two in average without loss of accuracy due to the smaller number of outputs of the energy layer. In most cases, the base learning rate is 0.001 for networks trained from scratch and 0.0001 for finetuning while the weight decay is 0.0005. Yet, this is not a fixed rule and even though it is relatively stable, the hyperparameters must be adjusted to the experiments (number of training samples, depth, finetuning, or from scratch). The results are also robust to small variations of the batch size yet it is also adapted to the training sizes. A batch size of 32 is used for the smallest training sets and up to 256 for ImageNet. Finally, input images larger than 227×227 are cropped to this size for the sake of comparison.

3.3 Datasets and experimental setups

The experiments are conducted on a total of ten datasets; seven are texture classification datasets, the other three are object recognition datasets.

The *ImageNet* 2012 dataset [50] contains 1,000 classes. The training set contains 1,281,167 images and the validation set used for testing 50,000 images (50 images/class) of size 256x256.
Three subsets of ImageNet are created in order to evaluate the domain transferability of features learned on texture and object image datasets to the classification of texture images. For each subset, 28 classes are selected from ImageNet and the training and testing splits of the full set are maintained. Therefore, each subset contains 1,400 test images and approximately 36,000 training images (35,513, 35,064, 36,318 for ImageNet-T, -S1 and -S2 respectively). The full list of classes selected for each subset is detailed in [25].

ImageNet-T is a subset which retains texture classes such as “stone wall”, “tile roof” and “velvet”. Based on visual examination, 28 classes with high texture content and with a single texture per image are selected.

[2]An implementation to train (from scratch and finetune) and test the T-CNN3 on kth-tips-2b is provided here: https://github.com/v-andrearczyk/caffe-TCNN

ImageNet-S1 is another subset with chosen object-like classes such as "Chihuahua", "ambulance" and "hammer". Based on visual examination, classes of object images with remarkable global shape are selected.

In the last subset *ImageNet-S2*, 28 classes are chosen randomly from the 1,000 classes.

kth-tips-2b [22, 112] contains 11 classes and 432 texture images per class. Each class is made of four samples, i.e. 108 images per sample. Each sample is used once for training while the remaining three samples are used for testing. The images are resized to 227×227 with nearest neighbour interpolation as for the resizing of the following datasets. The experiment is repeated ten times and the accuracy is averaged over the four splits and the ten repetitions. The standard deviation is also reported as the average over the four splits of the standard deviation of the accuracy over the ten repetitions.

Kylberg [31] is a texture database containing 28 classes of 160 images each of size 576×576. The validation setup is reproduced from [166]. One orientation out of 12 available is randomly chosen for each image. The images are split into four non-overlapping subimages which results in 17,920 images of size 288×288 which are resized to 256×256. A ten-fold cross-validation is used and the average accuracy is reported. The cross-validation folds are created once and kept fixed throughout the experiments for a fair comparison of the methods.

CUReT [113] is a texture database with 61 classes. The setup is reproduced from [99] in which 92 images are used per class, 46 for training, the other 46 for testing. The 200×200 images are resized to 227×227. The experiment is repeated 20 times and the average accuracy and standard deviation are reported.

DTD [99] consists of 47 classes which contain 120 images each obtained "in the wild". The images are of various sizes and even though using multiple input sizes is possible with the developed T-CNN, the images are resized to 227×227 for comparison with AlexNet which requires fixed input images. The dataset includes 10 available annotated splits with 40 training images, 40 validation images and 40 testing images for each class. The average accuracy and the standard deviation are reported over ten splits.

The *Macroscopic* [167] and *Microscopic forest species* databases [168] are used in Section 3.4.5 to test the developed CNNs on larger texture images (respectively $3,264 \times 2,448$ and $1,024 \times 768$). The Macroscopic dataset consists of 41 classes with over 50 images per class. Half of the set is used for training, the other half for testing. The Microscopic dataset contains 120 classes of 20 images each. In each class, 70% of the images are used for training, the rest for testing. The images of both datasets are resized to 640×640. These experimental setups are reproduced from [125] except that the results are averaged over ten trials instead of three, for more stable results.

Table 3.1: Classification accuracy (%) of various networks trained from scratch and finetuned (pre-trained on ImageNet). The number of trainable parameters (in millions) is indicated in brackets for 1,000 classes. The state of the art results are as reported by the authors in the original papers.

	Kylberg	CUReT	DTD	kth-tips-2b	ImNet-T	ImNet-S1	ImNet-S2	ImNet
				From scratch				
T-CNN1 (20.8)	89.5	97.0 ±1.0	20.6 ±1.4	45.7 ±0.5	42.7	34.9	42.1	13.2
T-CNN2 (22.1)	**99.2**	98.2 ±0.6	24.6 ±1.0	47.3 ±0.8	62.9	59.6	70.2	39.7
T-CNN3 (23.4)	**99.2**	98.1 ±1.0	**27.8 ±1.2**	**48.7 ±0.7**	**71.1**	**69.4**	**78.6**	51.2
T-CNN4 (24.7)	98.8	97.8 ±0.9	25.4 ±1.3	47.2 ±0.5	**71.1**	**69.4**	76.9	28.6
T-CNN5 (25.1)	98.1	97.1 ±1.2	19.1 ±1.8	45.9 ±1.3	65.8	54.7	72.1	24.6
AlexNet (60.9)	98.9	**98.7 ±0.6**	22.7 ±1.3	47.6 ±0.9	66.3	65.7	73.1	**57.1**
				Finetuned				
T-CNN1 (20.8)	96.7	99.0 ±0.3	33.2 ±1.1	61.4 ±0.5	51.2	46.2	53.5	-
T-CNN3 (23.4)	**99.4**	**99.5 ±0.4**	55.8 ±0.8	**73.2 ±0.6**	81.2	82.1	87.8	-
AlexNet (60.9)	**99.4**	99.4 ±0.4	**56.3 ±1**	71.5 ±1.2	**83.2**	**85.4**	**90.8**	-
state of the art	99.7 [166]	99.8 ±0.1[99]	75.5 ±1.1[123]	83.3 ±1.6[123]	-	-	-	-

3.4 Results and discussion

3.4.1 Networks from scratch and pre-trained

The results of the various T-CNN architectures (i.e. one to five convolution layers) trained from scratch and finetuned are reported in Table 3.1. The developed T-CNN3 outperforms AlexNet on most texture datasets: Kylberg, DTD, kth-tips-2b and ImageNet-T from scratch and Kylberg, CUReT, and kth-tips-2b in the finetuned experiments; while containing nearly three times fewer trainable parameters. Trained from scratch, T-CNN3 also performs well on the object-like and random subsets of ImageNet, i.e. ImageNet-S1 and ImageNet-S2. The T-CNN3 with simpler features and fewer parameters is also faster and easier to train on the smaller sub-datasets than AlexNet, which may explain why T-CNN3 outperforms AlexNet on these datasets from scratch. When using pre-trained networks, however, the original AlexNet outperforms the T-CNN architectures on non-texture images (i.e. ImageNet-S1 and ImageNet-S2), confirming intuition since it is designed for object recognition and can learn more complex and larger features from the large training set than the T-CNNs.

Note that the T-CNN does not compete with the state of the art as seen in Table 3.1 because the latter use much more complex and deeper architectures. For this reason, the comparison to AlexNet is more meaningful.

Table 3.2: Accuracy (%) of various network depths with average and maximum pooling of the energy layer.

	ImageNet-T		ImageNet-S1	
method	**average**	**max**	**average**	**max**
T-CNN1	42.7	41.3	34.9	24.1
T-CNN3	71.1	71.0	69.4	70.6
T-CNN5	65.8	67.4	54.7	67.6

3.4.2 Networks depth analysis

As reported in Table 3.1, the best T-CNN architecture is with three convolution layers (T-CNN3), while T-CNN4 and T-CNN2 are close behind. The energy layer measures the response to a set of patterns which are the results of the combination of multiple simple filters. Generally, the more convolution layers are used, the more complex the features sought in the last convolution layer are. In [49], the authors demonstrate that the concept of texture emerges at the first and second convolution layers, whereas high-level concepts such as objects and parts emerge in the deepest layers. This idea is confirmed in these experiments, as using five layers degrades the T-CNN performance since the fifth layer detects complex object-like features. Such large and complex features, unlike simpler local texture features, are likely to be sparsely detected in the input image, often detecting a single object in the image. Averaging the response to such features over the entire feature map seems inappropriate as it results in a massive loss of information. Intuitively, a maximum pooling of these deep features could achieve better results than an average pooling, especially in an object recognition task. To evaluate this idea, a maximum pooling layer is experimented instead of the average energy layer and is computed as follows:

$$e_j = \max(f_j(x_1, x_2)) \tag{3.3}$$

where x_1 and x_2 span the rows and columns of the input feature map f_j. This maximum pooling measures whether a certain feature is detected in the input image, disregarding its location and number of occurrences. The results in Table 3.2 confirm this idea, as the shallow T-CNNs are more accurate with an average energy layer while deeper ones are more accurate with a maximum pooling.

On the other extreme, features sought in the first layer are too simple (mainly edges and colours) to extract meaningful and discriminative information from the images. Firstly, the T-CNN1 obtains better results on texture datasets (ImageNet-T) and on the random selection of classes (ImageNet-S2) than on the object-like set (ImageNet-S1). The basic texture features extracted in the first convolution layer are not able to describe the complex object shapes which exhibit the most discriminative

Table 3.3: Classification results (accuracy %) on the kth-tips-2b dataset using networks pre-trained on different databases.

	ImageNet	ImageNet-T	ImageNet-S1	ImageNet-S2
T-CNN1	**61.4 ±0.6**	54.3 ±0.6	52.9 ±0.5	53.1 ±0.3
T-CNN3	**73.2 ±0.6**	61.8 ±1.0	56.3 ±0.5	59.0 ±0.5
AlexNet	**71.5 ±1.2**	56.9 ±0.4	55.5 ±0.7	58.2 ±0.6

information. Secondly, even in texture datasets, the accuracy of the T-CNN1 is significantly lower than T-CNN3 as these simpler and fewer features (fewer feature maps) cannot represent the more complex patterns present in the texture images. As shown in Table 3.1, T-CNN3 performs significantly better than T-CNN1 both from scratch and finetuned.

Note that this depth analysis does not generalise to all network architectures and datasets as a deeper approach (e.g. VGG-VD), with a large number of parameters, can implement complex functions and has obtained excellent results in texture classification with different texture descriptors (e.g. FV-CNN [8]). The relation between the depth of a network and the complexity and spatial support of the features varies between architectures. In deeper networks such as GoogleNet or VGG-VD, the receptive field and complexity of features increase slower throughout the network than in AlexNet. Thus, as shown in Section 3.4.7, the T-CNN approach can be successfully applied to deeper architectures.

3.4.3 Domain transferability

Domain transferability is a key concept of deep learning which enables obtaining high accuracy on many tasks with relatively little training data. As it is not possible to obtain labelled datasets like ImageNet for all recognition tasks, in particular for biomedical images, CNNs are often pre-trained on this large dataset. In this scenario, ImageNet is referred to as the source and the source task is object recognition. The pre-trained parameters are then finetuned on another dataset with a target task (in this case texture recognition) which can be close or far from the source task. The features learned on ImageNet are transferred to the target task and the weights of the last fully-connected layer are finetuned to remodel the combination of the detected features. Weights of previous layers are generally finetuned at a lower speed or kept unchanged. It is important to distinguish between the affine invariances required for general computer vision tasks (e.g. ImageNet) and some biomedical and texture datasets which only require invariances to rigid motions. For instance, the CNNs trained on ImageNet learned invariance to scale which may have a negative impact on datasets such as CUReT, Kylberg, forest species datasets and tissue images (Section 3.5) with fixed viewpoints and where scale can be a discriminative property. Yet, the networks largely benefit from the pre-training. A possible explanation for this observation is proposed in the following text. The invariance to scale can be learned independently from the recognition of patterns at different scales; i.e. two neurons respond to the same pattern at different scales and are combined in a deeper layer so that one deeper neuron responds to both scales. This is a (simplified) hierarchical property of CNNs in which deep representations are obtained from a composition of simpler and shallower ones. The weights can therefore be easily remodelled through finetuning, given sufficient training data, to drop the unnecessary invariances and learn important discriminations from the target dataset.

The accuracy of several networks pre-trained on multiple datasets and finetuned on a texture dataset are reported in Table 3.3. As expected, a network pre-trained on a texture dataset (ImageNet-T) achieves better results on another texture dataset than the same network pre-trained on a non-texture dataset (ImageNet-S1 and ImageNet-S2). This is explained as the tasks of object recognition and texture recognition are relatively distant. Yet, the amount of pre-training data predominantly influences the accuracy of the finetuned network in the target task. Indeed, all the networks pre-trained on the entire ImageNet significantly outperform the other ones. Note that the CNNs trained on an object detection task such as ImageNet sometimes learn to recognise parts on the image that one would not expect such as the grass or trees in the background which are highly textured. When such background is redundant within classes, the networks can learn this discriminant information even though it is not the salient object that is primarily sought. It shows that the network is able to learn texture-like patterns from texture regions (e.g. cheetah and background grass) and these features transfer particularly well to texture recognition tasks. These observations confirm that domain transferability greatly helps the training on texture datasets and suggest that a very large labelled texture dataset could bring a significant contribution to CNNs applied to texture analysis.

3.4.4 Visualisation

Visualising the features learned by T-CNNs from scratch and finetuned can provide an insight into the learning process as explained in Appendix A.4.7. The activation maximisation method via gradient ascent [169] is used in this section for visualisation. Examples of image patterns that activate the neurons in the third convolution layer (C3) are shown in Figure 3.2 for T-CNN3 trained on a texture dataset both from scratch and pre-trained on ImageNet. Similar examples for the last fully-connected layer (FC3) are depicted in Figure 3.3. The patterns which activate the neurons of the intermediate layer (C3) are more complex in finetuned networks than those trained from scratch. Note that in the finetuning approach, most of the learning is carried by the last fully-connected layer while the weights in intermediate layers vary only

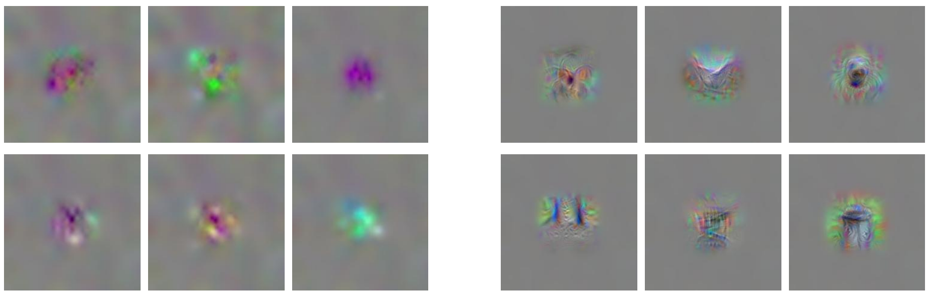

(a) from scratch **(b)** finetuned

Figure 3.2: Examples of activation maximisation of neurons in the third convolution layer. The T-CNN3 networks are trained on kth-tips-2b (a) from scratch and (b) finetuned (pre-trained on ImageNet). Figures obtained with the DeepVis toolbox [170].

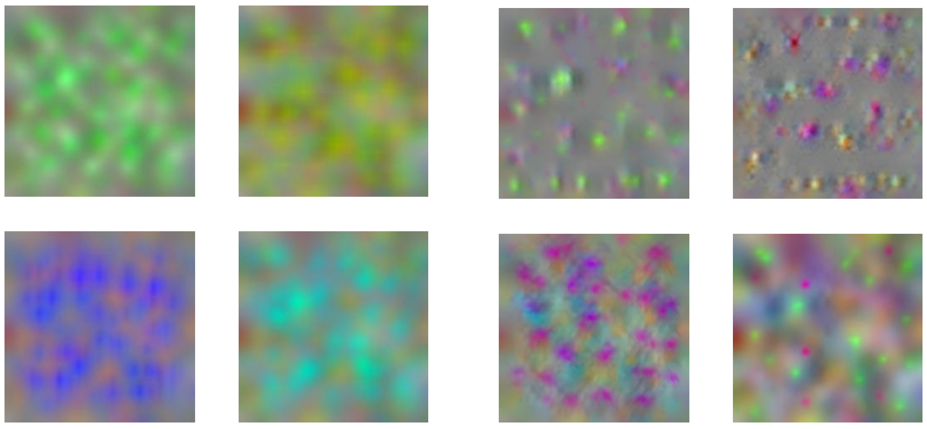

(a) from scratch **(b)** finetuned

Figure 3.3: Examples of activation maximisation of neurons in the last fully-connected layer (FC3). The T-CNN3 networks are trained on kth-tips-2b (a) from scratch and (b) finetuned (pre-trained on ImageNet). Figures obtained with the DeepVis toolbox [170].

slightly from the pre-trained weights as the learning is controlled by learning rates.

Therefore the features in Figure 3.2b are similar to the pre-trained ones before finetuning which explains their complexity. The complexity here refers to the shape of the patterns in the input image and their level of abstraction. For instance, the neurons in the third convolution layer of the finetuned network respond to complex patterns such as a head or an object as shown in Figure 3.2b. The features learned from scratch are simpler texture patterns. However, the features that activate the neurons of the last fully-connected layer (Figure 3.3) are of similar low complexity for both the networks from scratch and finetuned. These neurons mainly detect simple repeated patterns, even though the last neurons respond to patterns which are the combination of features in previous layers. Although the complexity generally increases with the depth, it is not the case here since the orderless pooling strategy of the energy layer discards the global shape information to focus on the repetition of simple texture patterns.

3.4.5 Results on larger images

The previous experiments were conducted on classic texture datasets with medium size images which fit most CNN input dimensions (AlexNet, VGG, GoogleNet, etc.). In this section, the developed method is evaluated on larger texture images from the forest species databases. The T-CNN approach, due to the average pooling strategy, enables using the same model architecture with various input sizes and transferring features between datasets of varying image sizes[3]. A fully convolutional approach [6] is not necessary here since the energy layer pools a single value per feature map regardless the input size. The weights of the T-CNN pre-trained on ImageNet can therefore be transferred and finetuned on the forest species images of size 640×640. The results obtained with T-CNN3 are compared to the state of the art in Table 3.4. Note that the standard AlexNet cannot be evaluated as it requires fixed input sizes (227×227). The T-CNN3 outperforms the state of the art [125] on the Macroscopic forest species dataset (+1.4%) and obtains similar results on the Microscopic one (-0.3%). Note that the method in [125] is of higher computational complexity and uses different hyperparameters for both datasets.

Table 3.4: Accuracy (%) of the T-CNN3 and comparison with the literature.

	Macroscopic	Microscopic
T-CNN3	**97.2** **±0.40**	97.0 ±0.61
[125]	95.77 ±0.27	**97.32** **±0.21**

3.4.6 Combining texture and shape analyses

A CNN architecture is developed to combine the texture and global shape analysis within a single network, referred to as Texture and Shape CNN (TS-CNN). Figure 3.4 illustrates this new architecture in which the energy layer of the T-CNN3 is extracted within the classic AlexNet. The output of the energy layer is concatenated to the flattened output of the last pooling layer, skipping the other convolution and pooling layers. In Caffe, existing layers are used to first flatten the outputs of the energy layer and of the last pooling layer P5 separately. These flattened outputs are then

[3]Note that this only holds to some extent as very small images may require an adapted architecture to take into account the size of the features and receptive fields as explained in Section 4.2. Alternatively, very large images with homogeneous properties and repetitive patterns may also require an adapted approach as shown in the application to tissue images in Section 3.5.

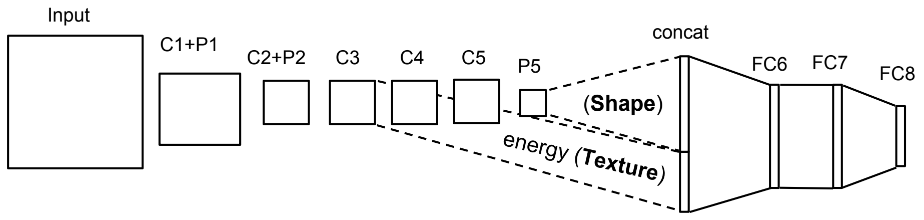


Figure 3.4: The architecture of the Texture and Shape CNN (TS-CNN-3), integrating T-CNN3 to a classic CNN (AlexNet).

Table 3.5: Classification results (accuracy %) on kth-tips-2b using AlexNet and T-CNN3 separately and combined as well as the state of the art method with a medium depth CNN (VGG-M). The number of trainable parameters in millions is indicated in brackets for 1,000 classes.

Shape	AlexNet (60.9)	71.5 ±0.9
	VGG-M FC-CNN [8]	71.0
Texture	T-CNN3 (23.4)	73.2 ±0.6
	VGG-M FV-CNN [8]	73.3
Texture and Shape	sum scores AlexNet T-CNN3 (84.3)	73.4 ±1.0
	TS-CNN3 (62.5)	**74.0** ±0.7
	VGG-M FV+FV-CNN [8]	73.9

concatenated and connected to the fully-connected layers without modifying the latter from the AlexNet layers. In this way, the features of the early layers (C1, C2 and C3) and weights of the fully-connected layers are shared by the texture and shape analyses, keeping the complexity of the network close to the classic AlexNet. Table 3.5 shows the improvements obtained with this texture and shape analysis. The networks are pre-trained on the ImageNet dataset and finetuned on kth-tips-2b.

In a first attempt to combine texture and shape, the classification vectors (outputs of the softmax layer) of T-CNN3 and AlexNet are summed to provide an averaged classification of the two networks. This simple score summing method achieves 73.4% accuracy, an increase of 0.2% as compared to the T-CNN3 and 1.9% as compared to AlexNet. In the second experiment, the network combining texture and shape analyses (TS-CNN) obtains the best result with 74.0%. These experiments show the complementarity of the texture and shape analyses on this dataset and the possibility to share the same simple features in early layers to conduct both analyses within the same network.

In Table 3.5, approaches within the "shape", "texture" and "texture and shape" method groups reveal comparable results. The TS-CNN is trained end-to-end as a single CNN architecture whereas [8] uses two CNNs of significantly higher com-

Table 3.6: Classification results (accuracy %) on the kth-tips-2b dataset using T-CNN based on GoogleNet.

	nb. layers	kth-tips-2b
T-CNN-G1	13 (2 FC)	75.6 ±0.7
T-CNN-G2	19 (2 FC)	**76.3 ±0.9**
GoogleNet	21 (1 FC)	75.0 ±0.8

plexity together with a FV encoding and an SVM classifier. Note that the number of parameters used in [8] is not reported in the table as this number differs between pre-training and finetuning and requires extra processing for encoding and classification. As a rough comparison, the VGG-M network has 101.7 million trainable parameters, whereas the T-CNN3 has 23.4 million and the TS-CNN3 62.5 million.

3.4.7 Deeper Texture CNN

The T-CNNs introduced so far have been based on the AlexNet architecture. The energy layer approach is now explored with the deeper GoogleNet Inception-v1 architecture [164] (see Appendix A.4.5). In GoogleNet, errors are measured at the end of the network, as well as after two intermediate inception modules in order to help the learning of parameters in early layers of the network. A new architecture is developed by adding energy layers before the fully-connected layers used to calculate the two intermediate errors. This architecture is pre-trained on ImageNet and is then finetuned on texture datasets. Three accuracies are provided at different depths; two at different intermediate layers after the energy layers and one at the end of the network. The first two accuracies can be obtained by finetuning shorter networks of reduced complexity similarly to the T-CNNs. For instance, the accuracy derived from the first energy layer after 11 convolution layers can be obtained from a shortened network which discards all the following convolution and inception blocks. The shortest network is referred to as T-CNN-G1 while the deeper one is referred to as T-CNN-G2. The full network is referred to as GoogleNet as it only differs from the original one by the energy layers in the derivation of the intermediate errors during pre-training. The T-CNN-G1 is 13 layers deep and the T-CNN-G2 19 layers deep, both including two fully-connected layers. Their complexity is significantly lower than the 21 layers GoogleNet as, with the width of the network increasing with depth, the deepest inception modules contain most parameters. The results of these deeper T-CNNs on kth-tips-2b and their depths are summarised in Table 3.6. These results demonstrate that the GoogleNet architecture also benefits from the orderless pooling of intermediate features with the energy layer, while significantly reducing the computational complexity of the network. Note that the original GoogleNet model, pre-trained on ImageNet and available from the Caffe zoo, achieves 74.9% accuracy on kth-tips-2b, almost equal to the 75.0% reported in Table 3.6.

3.4.8 Discussion

In this section, a new type of CNN architecture has been developed for analysing texture images. Inspired by classic neural networks and filter banks approaches, an energy measure has been introduced to discard the overall shape information analysed by classic CNNs. This method has achieved an increase of performance in texture recognition from two widely used networks (AlexNet and GoogleNet), while largely reducing the complexity, memory requirement and computation time. Finally, a network has been introduced to incorporate the developed texture specific approach into a classic network architecture, demonstrating their complementarity with an improvement of accuracy. The T-CNN architecture will be applied to biomedical tissue images in the next section.

3.5 Application to biomedical tissue images

3.5.1 Motivation

The analysis of tissue images is crucial for studying the cells behaviour and cellular senescence or detecting abnormal cells and cancers. The automatic analysis of these images helps to obtain more consistent, and more straightforward diagnoses. This section focuses on the recognition of malignant lymphomas and the classification of mouse liver tissue based on the age, gender and diet. The tissue images are obtained from biopsies sectioned and stained with Hematoxylin/Eosin (H&E). An application of the T-CNN is presented on this recognition task as an extension of the experiments in the previous section. As depicted in Figure 3.6, tissues exhibit high texture contents which make them good candidates for the proposed T-CNN approach. A framework is developed to use this low complexity network in an ensemble approach and significantly improves the state of the art on several tissue classification benchmarks.

3.5.2 State of the art

Recent methods used in the literature to which the developed framework will be compared are briefly described here. These approaches are based on classic machine learning techniques with hand-crafted feature extraction, dimensionality reduction, and classification.

A classification method named WND-CHARM (Weighted Neighbour Distance - Compound Hierarchy of Algorithms Representing Morphology) was developed in [171] and applied to many tissue images datasets. Its accuracy is reported with a hold-25%-out validation. In this method, a large set of statistical, wavelets and transform features are extracted from the image. The most discriminant features are classified with a variant of a nearest neighbour classifier. CP-CHARM (CellProfiler - Compound Hierarchy of Algorithms Representing Morphology) [172] is a more recent approach which extracts a large set of features which differs to some degree from the WND-CHARM features, reduces the dimension with PCA and uses linear discriminant analysis for classification. CP-CHARM is tested on the same datasets as WND-CHARM with both hold-25%-out and 10-fold cross-validation.

Texture and colour descriptors are evaluated in [173] both separately and jointly with an SVM classifier for the recognition of tissue images. Histograms in multiple colour spaces are used for the colour features while LBP and co-occurrence matrices are used as texture descriptors. In [174], a set of statistical and transform features are extracted from the images and classified with an ensemble SVM. This approach is applied to the prediction of mouse cells senescence by analysing liver tissue images.

3.5.3 Method

The number of labelled tissue images is often limited due to data privacy and to the need of experts for labelling the data. On the other hand, the resolution of microscopic tissue images being generally high, the images are large with commonly more than $1,000 \times 1,000$ pixels. Also, the analysed tissues exhibit highly repetitive texture patterns across the image. Therefore, the input images can be split to increase the number of training samples. A sum voting score can then be used at test time to combine the classification of the subimages in a collective decision. For this application, the T-CNN3 based on AlexNet is used. The proposed approach, named collective T-CNN, is illustrated in Figure 3.5 including the training and testing phases. The images of size $1,388 \times 1,040$ from the Image Informatics and Computational Biology Unit (IICBU) dataset [175] are split into 24 non-overlapping subimages as shown in Figure 3.5. The resulting images are resized to 227×227 with a nearest neighbour interpolation. In the training phase, all the subimages from the training set are used as independent samples to finetune the network pre-trained on ImageNet. In the testing phase, a sum voting score is applied among the 24 subimages to classify each full-size image. The collective score of a given test image is obtained by summing the softmax output vectors of the T-CNN3 for all the subimages. The obtained score vector gives for each class a confidence score, as a sum of 24 probabilities, to belong to that class. The original full-size image is assigned the class with the largest sum in the vector as follows:

$$c = \arg\max_i \sum_{n=1}^{24} \sigma^{(n)}[i] \tag{3.4}$$

where i spans the classes and $\sigma^{(n)}$ is the softmax output of the n^{th} subimage. This voting algorithm behaves as an ensemble model which takes a collective decision based on the analysis of different non-overlapping spatial areas of the image.

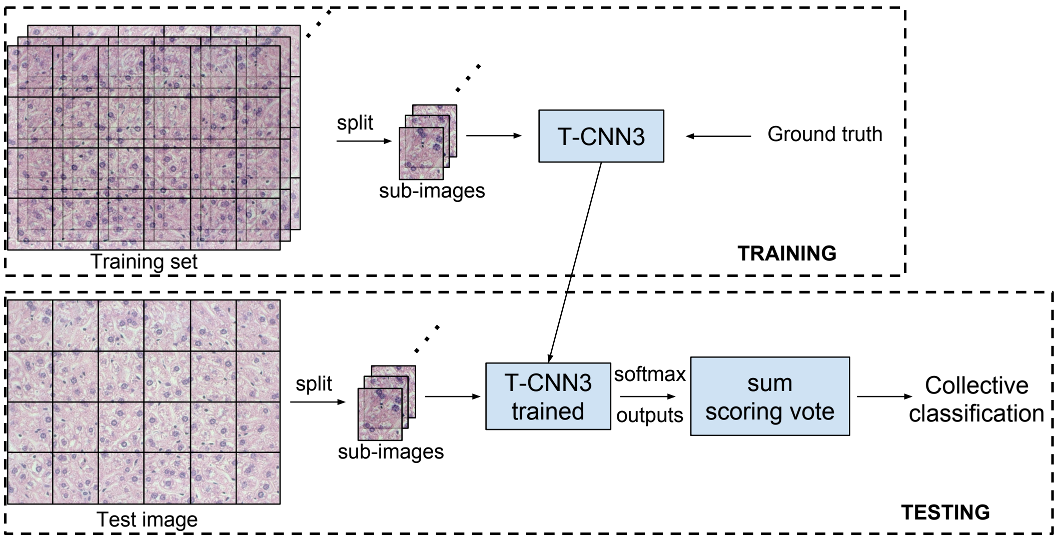


Figure 3.5: Training and testing phases of the developed collective T-CNN method with subimages and scoring vote.

3.5.4 Experiments

To compare with the state of the art, several experiments are reproduced from the literature. The datasets used in these experiments are part of the IICBU dataset [175], a collection of benchmarks for the analysis of biomedical images. It includes several benchmarks of organelles, cells, and tissues images available online for testing and evaluating automatic analysis algorithms.

AGEMAP

The Atlas of Gene Expression in Mouse Aging Project (AGEMAP) [175] contains images of livers from 48 male and female mice of four ages (1, 6, 16, and 24 months), on ad-libitum or caloric restriction diets. The sectioned livers were stained with H&E, and imaged by a bright-field microscope. All the staining and imaging were performed by the same person, which minimises the variability. The AGEMAP dataset is divided into four sub-datasets with a total of five validation setups.

The first sub-dataset and setup is the Liver Aging of female mice on Ad Libitum diet (**LA-female-AL**) [172] in which the aim is to predict the age of the mice. This benchmark contains 529 images from 11 mice grouped into four classes (1, 6, 16 and 24 months). The validation process is a 10-fold cross-validation repeated 100 times. The accuracy is averaged over the 10 folds for all 100 iterations and the median and standard deviation across the 100 iterations is reported.

The second sub-dataset and setup is the Liver Aging Across Subjects on Ad Libitum diet (**LA-AS-AL**) reproduced from [174]. It contains four classes (1, 5, 16 and 24 months) and 1,027 images from 21 mice. For each mouse, all the images are randomly divided into training (5/6) and test images (1/6). The accuracy is averaged over 30 runs and reported together with the standard deviation.

The third sub-dataset: Liver Gender 6 Months on Ad Libitum diet is used in [173, 172] in which the aim is to predict the gender of the mice in a binary classification. It contains two classes (male and female) with images from six mice for a total of 265 images. Note that two different validation setups are used with this sub-dataset as follows:

The third setup: (**LG6M-AL-5p**) is reproduced from [173], as a hold-95%-out validation, i.e. 5% of the data is used for training, the rest for testing. The reason for this split is to experiment with few training samples to simulate real conditions. The Mean Average Precision (MAP) measure is averaged over 5,000 runs as suggested in [173].

The fourth setup: (**LG6M-AL**) is reproduced from [172]. It is the same validation setup as the one used in LA-female-AL with a 10-fold cross-validation repeated 100 times.

The last sub-dataset and setup derived from the AGEMAP dataset is the Liver Gender 6 Months on Calories Restriction diet (**LG6M-CR**) reproduced from [172]. This sub-dataset contains two classes (male and female) with a total of 303 images obtained from six mice. The validation setup is the same as LA-female-AL and LG6M-AL with a 10-fold cross-validation repeated 100 times.

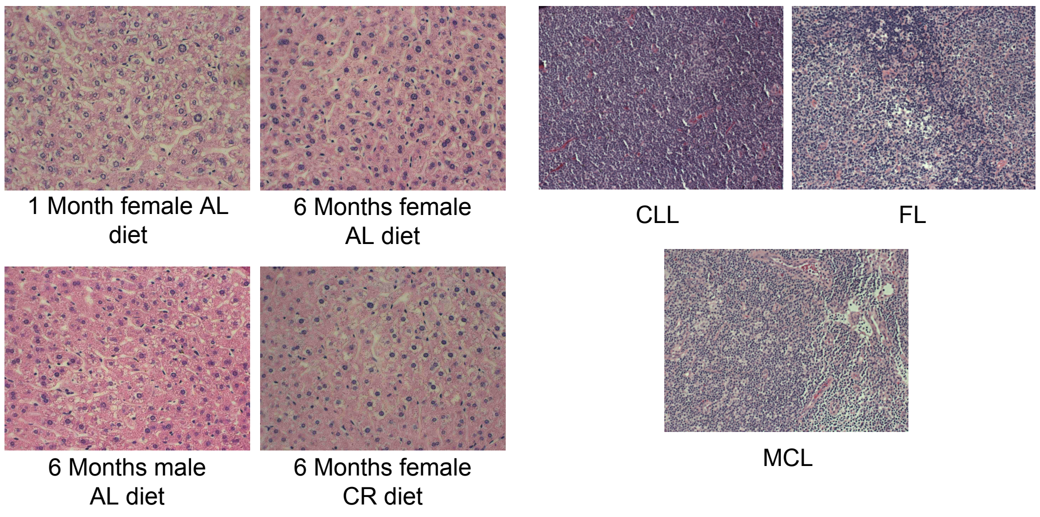
Lymphoma

The Lymphoma dataset [175] contains images of three types of malignant lymphoma, namely Chronic Lymphocytic Leukemia (CLL), Follicular Lymphoma (FL), and Mantle Cell Lymphoma (MCL). Lymphoma is a cancer affecting the lymph nodes. Its detection and diagnosis can greatly benefit from the automatic analysis of tissue images obtained from biopsies stained with H&E. In particular, this dataset was developed to evaluate methods to automatically distinguish the three types of lymphoma. The samples were prepared by different pathologists at different sites, leading to a large variation in staining and image quality which reflects real acquisition conditions. It also makes the automatic classification a realistic and challenging task. The Lymphoma dataset contains 374 images grouped into three classes (CLL, FL and MCL). It is used with two different validation setups as follows:

The first setup (**Lymphoma-5p**) is reproduced from [173]. It is the same validation setup as LG6M-AL-5p with 5% of the data used for training, the rest for testing. The MAP is averaged over 5,000 runs.

The second setup: (**Lymphoma**) is reproduced from [172]. The validation setup is the same as the one used in LA-female-AL, LG6M-AL, and LG6M-CR with the repeated 10-fold cross-validation.

To summarise, two datasets (AGEMAP and Lymphoma) are divided into five different sub-datasets for a total of seven experiments with different validation setups. Example images of the AGEMAP and Lymphoma datasets are shown in Figure 3.6.



(a) AGEMAP **(b)** Lymphoma

Figure 3.6: Examples of H&E stained tissue images from the IICBU dataset [175] (a) mice tissue liver images (AGEMAP) and (b) Lymphoma tissue images.

3.5.5 Results

The results on the 10-fold cross-validation setups are presented in Table 3.7. Note that the results of WND-CHARM are given in [172] and are obtained from a hold-25%-out validation. Yet, they are compared to the other results as it was shown in [172] that the average accuracy is similar to a 10-fold cross-validation although the results vary more between the tests (higher standard deviation). The collective T-CNN developed in this section obtains significantly better results with an increase of accuracy of up to 32.7% on the Lymphoma dataset. This large difference in accuracy must be put into perspective as CP-CHARM and WND-CHARM use simple machine learning methods and are generalised image classifiers, not specific to texture analysis. They are not designed for a specific task and perform reasonably well on many various tasks, whereas the collective T-CNN uses deep learning and is specifically designed for the analysis of texture tissue images.

The developed collective T-CNN achieves 100% accuracy on all but one of the 10-fold cross-validation experiments which demonstrates its accuracy and reliability. To evaluate the benefit of the collective classification with sum scoring vote, it can be compared with the average accuracy of all subimages. The average per-subimage classification accuracy is 99.7%, 98.2%, 95.2% and 95.5% for respectively LA-female-AL, LG6M-AL, LG6M-CR, and Lymphoma. These results demonstrate the important impact of the collective decision on the accuracy of the method.

Table 3.7: Classification accuracy (%) of the collective T-CNN and comparison with the state of the art on the 10-fold cross-validation setups. The WND-CHARM results are obtained from a hold-25%-out validation.

	LA-fem-AL	LG6M-AL	LG6M-CR	Lymphoma
Collective T-CNN	**100** ±0.14	**100** ±0.0	**100** ±0.0	**98.7** ±0.33
CP-CHARM [172]	89 ±0.4	98 ±0.5	99 ±0.1	66 ±0.1
WND-CHARM [171]	93 ±3	98 ±1	99 ±1	79 ±4

Table 3.8: Classification accuracy (%) and Mean Average Precision (MAP) (%) of the collective T-CNN and comparison with the state of the art on other validation setups.

	LA-AL-AS	LG6M-AL-5p (MAP)	Lymphoma-5p (MAP)
Collective T-CNN	**99.1** ±0.7	**98.7** ±3.2	**65.1** ±5.7
feature selection + SVM [174]	97.01	-	-
$LBP_{riu}+I1H2H3$ [173]	-	97.3	57.6
CCOM [173]	-	82.5	63.3

The results on LA-AL-AS, LG6M-AL-5p and Lymphoma-5p are presented in Table 3.8. The developed approach again outperforms the state of the art methods on the three sub-datasets. Of all the experiments, only Lymphoma-5p seems to create problems for the collective T-CNN with 65.1% accuracy. This lower accuracy can be explained by the variation in data acquisition and the low number of training images. The confusion matrices for these datasets are shown in Tables 3.9, 3.10 and 3.11. The confusion matrix of Lymphoma-5p shows that most confusions occur between the CLL and MCL classes. Table 3.11 illustrates the correlation between the age of the mice and the misclassification as the few confusions occur between images of mice of close age, e.g. 6 months misclassified as 1 month and 16 months. Confirming intuition, the similarity between the classes decreases as the difference between the ages increases.

Table 3.9: Confusion matrix of the collective T-CNN on Lymphoma-5p.

	True		
	CLL	FL	MCL
CLL	0.56	0.16	0.28
FL	0.1	0.75	0.14
MCL	0.26	0.21	0.53

Table 3.10: Confusion matrix of the collective T-CNN on LG6M-AL-5p.

	True	
	Male	Female
Male	0.93	0.07
Female	0.06	0.94

Table 3.11: Confusion matrix of the collective T-CNN on LA-AL-AS.

	True			
	1 Month	6 Months	16 Months	24 Months
1 Month	0.99	0.01	0	0
6 Months	0	0.99	0.01	0
16 Months	0	0.02	0.98	0
24 Months	0	0	0	1

The average per-subimage classification accuracy for the LA-AL-AS sub-dataset is 93.1% vs. 99.1% with the collective decision, which again demonstrates the power of the ensemble approach in the final classification.

Finally, similar observations are made on the visualisation of the features learned by the network as in Section 3.4.4. The neurons in the last fully-connected layer respond to simple repeated texture patterns.

3.5.6 Discussion

An application to tissue image classification has been presented based on the analysis of non-overlapping areas of the images using the T-CNN3 and a scoring vote method. The texture in tissue images being repetitive, each image patch can be treated as an independent sample at training time and the predictions of multiple patches can be combined at test time. While labelled training sets for biomedical imaging are generally small, the developed approach can benefit from the high resolution images by increasing the number of training samples with this splitting approach. After training, the predictions of subimages are combined to accurately classify an unknown image. The collective T-CNN has largely improved the state of the art on multiple tissue images benchmarks including microscopic mice liver tissues and lymphoma tissue images.

Chapter 4

Dynamic texture recognition with convolutional networks

4.1 Introduction

This chapter presents a new framework based on convolutional networks for the classification of DTs. The idea is to extend the T-CNN introduced in Chapter 3 to the analysis of DTs based on the spatial distribution of pixels and on their evolution and dynamics over time. To this end, T-CNNs are trained to analyse the sequences on three orthogonal planes in the $2D + time$ space. This approach is partly inspired by the extension of LBP on three orthogonal planes (LBP-TOP) introduced in [14]. It is also influenced by ensemble models which extract useful diverse knowledge from the training data by learning different models in parallel and by averaging their predictions. An overview of the new approach, referred to as Dynamic Texture CNN (DT-CNN), is illustrated in Figure 4.1 and explained in more detail in Section 4.2.2. Slices from the DT sequences are first extracted to train an independent T-CNN on each plane. The outputs of all the slices on the three planes are then summed to obtain the class with maximum prediction score during the test phase in an ensemble data fusion approach.

The main contributions of this chapter are described as follows:
(1) A new framework is introduced to analyse DTs on three orthogonal planes;
(2) networks adapted to the small images of several DT datasets are developed based on the original T-CNN;
(3) experiments are conducted on three DT databases using seven benchmarks with considerable differences in terms of number of classes, inter- and intra-class variation number and size of images, and experimental protocol;
(4) an evaluation of the contribution and complementarity of each plane is conducted as well as an analysis of domain transferability of trained parameters.

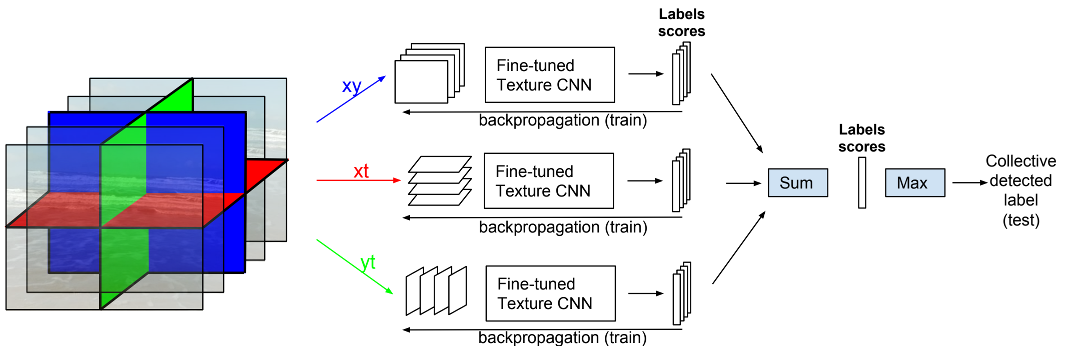


Figure 4.1: An overview of the proposed DT-CNN for the classification of a DT sequence based on T-CNNs on three orthogonal planes in an ensemble model approach. The T-CNNs separately classify slices extracted from three planes of a DT sequence. The outputs of the last fully-connected layers are summed and the highest score gives the collective classification decision.

The rest of this chapter is organised as follows: The new method for DT analysis with T-CNNs on three orthogonal planes is introduced in Section 4.2. The experimental setups and datasets on which the developed method is evaluated are presented in Section 4.3. The results are described in Section 4.4 and compared with the state of the art. Finally, a discussion is proposed in Section 4.5.

4.2 Materials and Methods

The main idea of the proposed DT-CNN is to use convolutional networks on three orthogonal planes in order to learn, pool and classify features which are repetitive in the spatial and temporal domains.

4.2.1 Texture CNN

The texture specific CNN was introduced in the previous chapter. In this chapter, the T-CNN3 based on AlexNet and the T-CNN-G2 based on GoogleNet are used.

A new version of these networks is also developed for smaller input images. A CNN is used in [160] to analyse small DT sequences (frame size: 50×50). The authors dilate and warp the input images in order to match the input frame size of AlexNet (227×227). This approach is not optimal both in terms of complexity of the network and its capability to learn from the small images. Instead, new architectures are designed in this chapter to analyse small input images such as those of the Dyntex++ (frame size: 50×50) and UCLA (frame size: 48×48) datasets. While the T-CNNs do not require fixed input sizes due to the energy layer, it is necessary to adapt them to these small input sizes for the following reasons. The receptive field of the neurons should be small enough to tile the input image in such a way that the

Table 4.1: Architectures of the T-CNN3 and T-CNN3-S based on AlexNet, where c is the number of colour channels and N is the number of classes.

	Output sizes		**kernel, pad, stride**		**trainable param.**	
Layer type	T-CNN3	T-CNN3-S	T-CNN3	T-CNN3-S	T-CNN3	T-CNN3-S
crop	$c \times 48 \times 48$	$c \times 227 \times 227$	-	-	0	0
Conv (C1)	$96 \times 48 \times 48$	$96 \times 55 \times 55$	5, 2, 1	11, 0, 4	$c \times 2{,}400 + 96$	$c \times 11{,}616 + 96$
ReLU	$96 \times 48 \times 48$	$96 \times 55 \times 55$	-	-	0	0
Pool (P1)	$96 \times 24 \times 24$	$96 \times 27 \times 27$	2, 0, 2	3, 0, 2	0	0
LRN	$96 \times 24 \times 24$	$96 \times 27 \times 27$	-	-	0	0
Conv (C2)	$256 \times 24 \times 24$	$256 \times 27 \times 27$	3, 1, 1	5, 2, 1	$221{,}440$	$614{,}656$
ReLU	$256 \times 24 \times 24$	$256 \times 27 \times 27$	-	-	0	0
Pool (P2)	$256 \times 12 \times 12$	$256 \times 13 \times 13$	2, 0, 2	3, 0, 2	0	0
LRN	$256 \times 12 \times 12$	$256 \times 13 \times 13$	-	-	0	0
Conv (C3)	$384 \times 12 \times 12$	$384 \times 13 \times 13$	3, 1, 1	3, 1, 1	$885{,}120$	$885{,}120$
ReLU	$384 \times 12 \times 12$	$384 \times 13 \times 13$	-	-	0	0
Energy	384	384	-	-	0	0
Fully-con. (FC1)	$3{,}000$	$4{,}096$	-	-	$1{,}155{,}000$	$1{,}576{,}960$
ReLU	$3{,}000$	$4{,}096$	-	-	0	0
Dropout	$3{,}000$	$4{,}096$	-	-	0	0
Fully-con. (FC2)	$3{,}000$	$4{,}096$	-	-	$9{,}003{,}000$	$16{,}781{,}312$
ReLU	$3{,}000$	$4{,}096$	-	-	0	0
Dropout	$3{,}000$	$4{,}096$	-	-	0	0
Fully-con. (FC3)	N	N	-	-	$3{,}000 \times N + N$	$4{,}096 \times N + N$
Softmax	N	N	-	-	0	0

energy layer will behave similarly to a filter bank spanning the input image. Using a T-CNN3 architecture, the receptive field of the neurons in the third convolution layer would be larger than the input image itself. The new network, referred to as T-CNN3 Small (T-CNN3-S), is detailed and compared to the T-CNN3 in Table 4.1. Similarly, a version of T-CNN-G2 is developed for small images.

As mentioned in Appendix A and Section 3.4.3, the domain transferability of CNNs is an important aspect for training deep architectures. The kernels learned by any network can generally be transferred to another network if the kernel sizes are equal, including a CNN with smaller input size. However, the kernel sizes of the T-CNN3 and the T-CNN3-S being different, the latter cannot be initialised using the kernels learned by the T-CNN3 on ImageNet [50]. Therefore, the T-CNN3-S and its equivalent based on GoogleNet are pre-trained on a version of ImageNet with images resized to 50×50.

The resulting DT recognition frameworks are referred to as DT-AlexNet and DT-GoogleNet depending on the network architecture[1].

[1]The normal and reduced architectures (e.g. T-CNN3 and T-CNN3-S) are reported as the same DT framework for simplicity (e.g. DT-AlexNet).

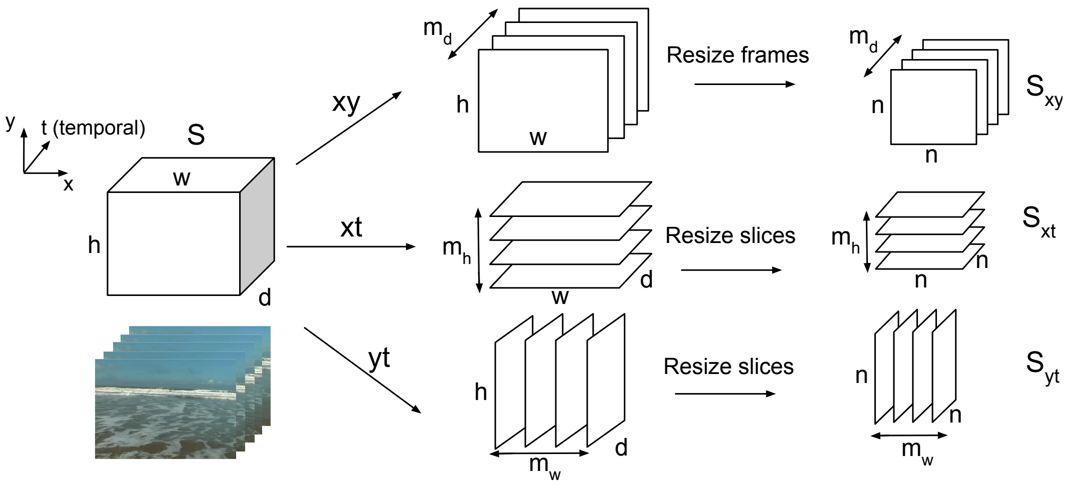


Figure 4.2: A diagram of the DT sequence slicing in three orthogonal planes.

4.2.2 Dynamic Texture CNN

Slicing the Dynamic Texture data

Slices of the DT sequences are extracted as illustrated in Figure 4.2 to enable the training of the networks on the three orthogonal planes.

XY plane (spatial): A sequence of DT with d frames of size $h \times w$ is represented as $S \in \mathbb{R}^{h\times w\times d\times c}$ where h (height), w (width) and d (depth) are in the x, y, and t axes respectively and c is the number of colour channels, i.e. three for RGB or one for greyscale. In the spatial plane, m_d frames equally spaced in the temporal axis are extracted from S. All the frames are resized using bilinear interpolation to the size $n \times n$ to obtain a sequence $S_{xy} \in \mathbb{R}^{n\times n\times m_d\times c}$ with $m_d \leq \min(d,h,w)$ and $n \leq \min(d,h,w)$.

XT and YT planes (temporal): From the same sequence S, m_h and m_w slices are extracted in the xt and yt planes, equally spaced on the y and x axes respectively. The slices are resized to $n \times n$ resulting in sequences $S_{xt} \in \mathbb{R}^{n\times n\times m_h\times c}$ and $S_{yt} \in \mathbb{R}^{n\times n\times m_w\times c}$. A slice in the xt (or yt) plane reflects the evolution of a row (or a column) of pixels over time throughout the sequence. After pre-processing a sequence, three sets of slices are obtained which represent the same DT in three different planes. Examples of spatial and temporal slices are shown in Figure 4.3.

Training on three planes

In order to evaluate the developed methods, the sequences are split into training and testing sets. Details on the training and testing splits are provided in the experimental setups in Section 4.3.1. A dataset containing M original sequences is split into T training sequences and $(M-T)$ testing sequences. In each plane, there is a total of $T \times m$ training and $(M-T) \times m$ testing slices, where $m \in \{m_d, m_h, m_w\}$ is the number of slices per sequence. For each plane, the $T \times m$ training slices are used to finetune an independent network. In the testing phase, the slices in each plane are classified and the outputs are combined as explained in the following section.

Sum collective score

An independent network is used for each of the three orthogonal planes, thus multiple outputs must be combined in the testing phase. A collective score is implemented by summing the output predictions of the T-CNNs. Firstly, a sequence is represented in each plane by a stack of slices. Therefore, a score for a given plane is obtained by summing the outputs of all the slices in this plane. The score vector of a sequence in a plane p with m slices is computed as follows:

$$\mathbf{s}^p = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_i^p \tag{4.1}$$

where $\mathbf{s}_i^p \in \mathbb{R}^N$ is the output (non-normalised classification score) of the last fully-connected layer of the i^{th} slice on plane p, with $p \in \{xy, xt, yt\}$ and N is the number of classes. All the scores $\mathbf{s}_i^p$ with $i = \{1, ..., m\}$ are obtained with the same finetuned network for a particular plane p and each plane uses an independently finetuned network. A global score for a given sequence is then obtained by summing over the three planes as follows:

$$\mathbf{s} = \sum_{p=\{xy,xt,yt\}} \mathbf{s}^p \tag{4.2}$$

Note that in Section 4.4.2, sums over two planes or single planes are also used to analyse their contribution and complementarity.
The collectively detected label l for a sequence is the one for which the sum score $\mathbf{s}$ is maximum.

$$l = \arg\max_j (\mathbf{s}[j]) \tag{4.3}$$

where $j = \{1, ..., N\}$ enumerates the DT classes. This ensemble model approach combines three weak classifiers to create a more accurate one. A late data fusion approach is used as it requires three network classifiers to recognise three data types derived from the sequences, i.e. slices in three different planes. The late fusion adopted here is different from the one used in [155], in which the fully-connected layers combine multiple "streams" of frames analysed at multiple time steps. By using a sum collective score, the classification confidence of each slice, given by the output vector of the last fully-connected layer, is taken into account for the collective classification. Confidence in this context refers to the magnitude of the output activations of the convolutional network as each neuron gives a score similar to a non-normalised probability for the input image to belong to a certain class.

Moreover, summing the raw output of the last fully-connected layer gives better results than the softmax normalised probability output. Using the raw output, large non-normalised scores can be attributed to a sequence by a single plane for a particular class if the confidence is high. This is similar to an automatic weighting strategy based on the detection confidence of each network. Note that in Section 3.5, the softmax outputs are summed. Although there is no large difference in using the softmax or raw outputs, the reasoning is that the subimages are from the same image and same plane. Due to the homogeneity of tissue images, all subimages are expected to contribute similarly to the collective score. On the other hand, summing across multiple planes means that one plane could be detected with more confidence than the others and this confidence should have a greater impact. It may also help by weighting slices in sequences which are not repetitive throughout the entire temporal domain.

Finally, it was confirmed experimentally that a sum collective score performs better than a majority or a Borda count voting scheme.

4.2.3 Domain transfer

Similarly to the previous chapter, the networks are pre-trained on ImageNet to transfer the knowledge learned from this large image dataset to the DT analysis. It is only possible to pre-train on the spatial (*xy*) plane since ImageNet does not contain videos, yet it transfers relatively well to the analysis of temporal slices (*xt* and *yt*). As suggested in [176], transferring features from the relatively distant task of object recognition is better than using random initialisations for the recognition of temporal texture slices. Although the improvement in terms of accuracy is minor, using pre-trained networks also makes the training significantly faster.

Several video datasets exist such as human actions and sports recognition [155]. The proposed approach, however, is designed for the classification of homogeneous and repetitive DTs as opposed to activities localised in space and motion events localised in space and time [23]. The temporal slices are extracted at multiple spatial locations of the sequence and should all exhibit the dynamic of the analysed DT. By definition, the spatial slices should also exhibit the same texture properties across time. Therefore, the networks are not pre-trained on other video datasets as sequences localised in space and/or time would extract multiple types of dynamic with potentially only a few slices that represent the dynamic of interest.

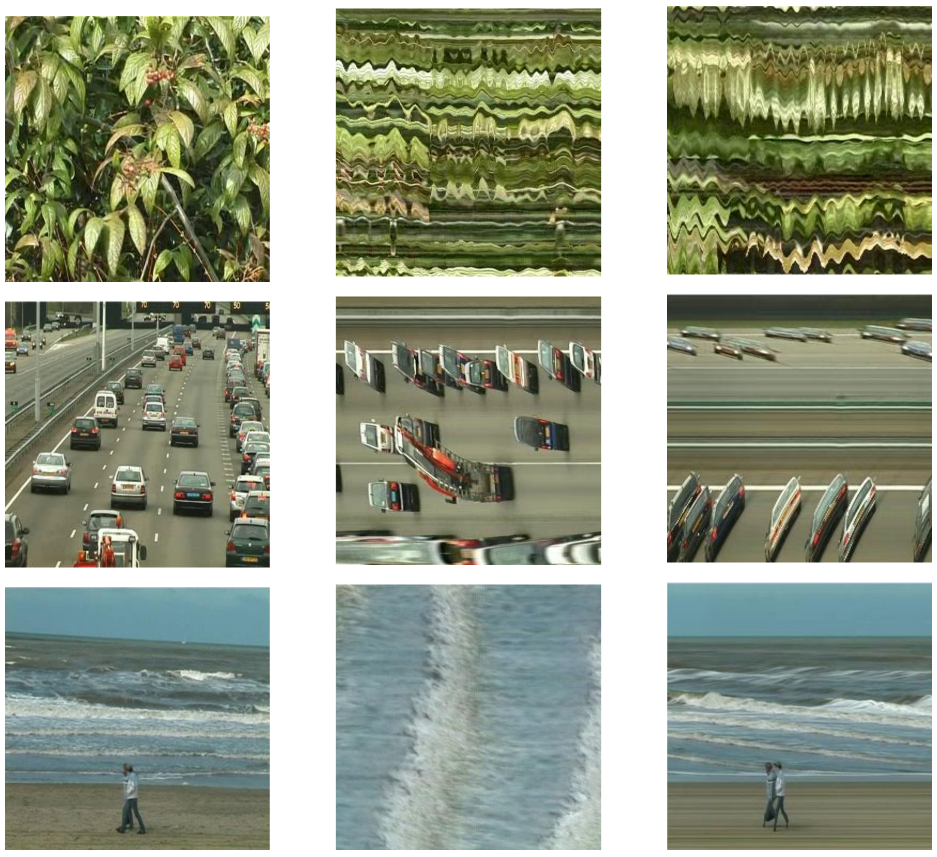


Figure 4.3: Examples of DT slices in three orthogonal planes of foliage, traffic and sea sequences from the DynTex database. (a) *xy* (spatial), (b) *xt* (temporal) and (c) *yt* (temporal).

4.3 Datasets and experimental setups

This section presents the datasets and protocols used in the experiments and provides implementation details.

4.3.1 Datasets

Three datasets are used and organised in seven benchmarks to test the developed algorithm and compare the results with the state of the art. These experiments cover most of the DT datasets used in the literature. They include many DT types, various number of training samples, balanced and unbalanced classes, static and moving cameras, rotation, illumination and scale variation as well as several validation setups (leave-one-out, N-folds cross-validation, and random splits).

The DynTex Database [24] is a diverse collection of DT videos. Three sub-datasets are compiled from the DynTex database. For all of them, the processed and downsampled (352×288) colour images are used. All sequences are at least 250 frames long, therefore 250 frames are used for all sequences to extract the slices. The experimental setups are reproduced from [41] with a leave-one-out classification and the average classification accuracies are reported. The three sub-datasets are defined as follows:

DynTex alpha contains 60 DT sequences equally divided into three categories: “sea” (20), “grass” (20) and “trees” (20), where the number of sequences is given in brackets for each class.

DynTex beta contains 162 sequences divided into 10 classes: “sea” (20), “vegetation” (20), “trees” (20), “flags” (20), “calm water” (20), “fountains” (20), “smoke” (16), “escalator” (7), “traffic” (9) and “rotation” (10).

The *DynTex gamma* dataset [24] contains 264 sequences grouped into 10 classes: “flowers” (29), “sea” (38), “naked trees” (25), “foliage” (35), “escalator” (7), “calm water” (30), “flags” (31), “grass” (23), “traffic” (9) and “fountains” (37).

Dyntex++ [39] is derived from the DynTex database by cropping and processing the sequences. It consists of 36 classes of 100 sequences each. Each sequence contains 50 greyscale frames of size 50×50. The experimental setup is reproduced from [39], where 50 sequences are randomly selected from each class as the training set, and the other 50 sequences are used for testing. This process is repeated 20 times and the average classification rate is reported.

The UCLA database [45] contains 50 classes of four DT sequences each. A DT sequence includes 75 greyscale frames of size 160×110 which are cropped to the size 48×48 to show representative dynamics. The classes can be grouped together to form more challenging sub-datasets. Three commonly used setups are evaluated, defined as follows:

UCLA 50-class: This setup is reproduced from [39]. A 4-fold cross-validation is performed with three sequences of each class used for training, the remaining one for testing in each of the four splits (i.e. 150 training and 50 testing sequences).

UCLA 9-class: This experimental setup is also reproduced from [39]. The sequences taken from different viewpoints are grouped into 9 classes: “boiling water” (8), “fire” (8), “flowers” (12), “fountains” (20), “plants” (108), “sea” (12), “smoke” (4), “water” (12) and “waterfall” (16), where the number of sequences is given in brackets for each class. In each class, 50% of the sequences are used for training, the other 50% for testing (i.e. 100 training and 100 testing sequences). The average classification rate is reported over 20 trials with random splits which are created once and used unchanged throughout the experiments.

UCLA 8-class is similar to the 9-class setup except that the “plant” category is removed since it contains many more sequences than the other classes. The number of remaining sequences is 92, which is split into 46 training and 46 testing samples.

The same evaluation is used as in the 9-class setup with 20 trials. This experimental setup is reproduced from [177].

Table 4.2: Hyperparameters used for training the T-CNNs on different datasets. From left to right: initial learning rate, factor gamma by which the learning rate is multiplied at every step, weight decay, momentum, batch size, number of iterations and steps.

hyperparam.	lr	γ	weight decay	momentum	batch size	nb. iter	steps
Dyn++	0.01	0.01	0.0005	0.9	64	25,000	5,000; 20,000
UCLA-9	0.01	0.01	0.0005	0.9	64	4,000	1,000; 3,000
Dyn, UCLA-50, UCLA-8	0.0001	0.1	0.004	0.9	64	2,000	1,500

4.3.2 Implementation details

The networks are implemented with Caffe[2] [165]. As explained in Section 4.2.1, different architectures are developed to adapt the T-CNNs to the large difference of image sizes in the various experiments. The results are reported without distinction since the same approach is used for all the experiments as described in Section 4.2.2. Two methods are reported, namely DT-AlexNet and DT-GoogleNet depending on the T-CNN architecture employed.

The hyperparameters of the networks slightly depend on the size of the datasets, setups and input sizes and are summarised in Table 4.2. The number of slices per sequence is equal in the three planes and the results are relatively robust to variations of this number. For Dyntex++ and UCLA, the number of slices is equal to the height, width, and depth of the sequences, i.e. 50 and 48 respectively. The sequences of the DynTex database being much larger ($352 \times 288 \times 250$), only 10 slices are extracted per plane. All the slices from DynTex are resized after extraction to 227×227. The networks are trained using stochastic gradient descent with softmax and multinomial logistic loss.

4.4 Results and discussion

4.4.1 Results

The results of the proposed DT-CNN approach are compared with the state of the art in Table 4.3. The methods from the literature that obtain the best results on each dataset are reported, namely 9-Plane mask Directional Number transitional Graph with SVM classifier (DNG) [152], Dynamic Fractal Spectrum (DFS) [177], spatial Transferred ConvNet Features (s-TCoF) and concatenation of spatial and

[2]An implementation of the method is available at the following GitHub repository: https://github.com/v-andrearczyk/DT-CNN

Table 4.3: Accuracy results (%) of the proposed DT-CNN approaches and of the state of the art on multiple DT datasets.

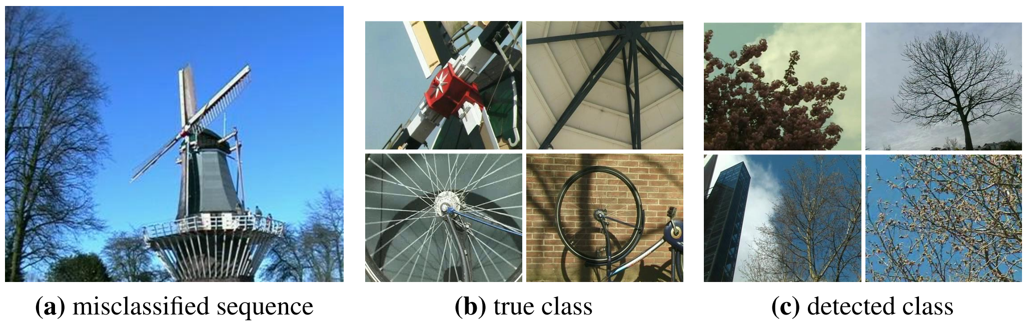
	Dyntex++	Dyn-alpha	Dyn-beta	Dyn-gamma	UCLA-50	UCLA-9	UCLA-8
LBP-TOP [14, 178, 41]	71.2	96.67	85.8	84.85	86.1	-	96.8
DNG [152]	93.8	-	-	-	-	**99.6**	**99.4**
DFS [177]	89.9	-	-	-	**100**	97.5	99
s-TCoF [41]	-	**100**	99.38	95.83	-	-	-
st-TCoF [41]	-	98.33	98.15	98.11	-	-	-
MEWLSP [137]	98.48	-	-	-	96.5	98.55	98.04
DT-AlexNet	98.18	**100**	99.38	**99.62**	99.5	98.05	98.48
DT-GoogleNet	**98.58**	**100**	**100**	**99.62**	99.5	98.35	99.02

temporal Transferred ConvNet Features (st-TCoF) [41] and Multiresolution Edge Weighted Local Structure Pattern (MEWLSP) [137]. Finally, the LBP-TOP [14] is also reported as it was tested on most of the datasets and also combines an analysis of three orthogonal planes. The results of the LBP-TOP are provided in [178] and [41] with the original author's implementation [14] for the former and their own implementation for the latter. The proposed DT-CNN methods (DT-AlexNet and DT-GoogleNet) consistently obtain high accuracy results on all the datasets and overall outperform all other methods in the literature. The UCLA dataset contains few training samples per class and therefore the DT-CNNs do not outperform the current state of the art, but a significant improvement is obtained on the large DynTex and Dyntex++ datasets. The consistency of high accuracy results across all the datasets demonstrates the robustness of the developed approach. The deep DT-GoogleNet approach outperforms the shallower DT-AlexNet even though more challenging datasets would be required to fully demonstrate the power of using deep architectures. In the following paragraphs, the results of each experiment are discussed in more detail.

DynTex alpha: The classification rates of DT-AlexNet and DT-GoogleNet are 100% on this dataset which is not challenging as it contains high inter-class separability and low intra-class variability and each sequence has at least one other sequence of the same class which is very similar. The spatial TCoF (s-TCoF) in [41] also obtains 100% on this dataset.

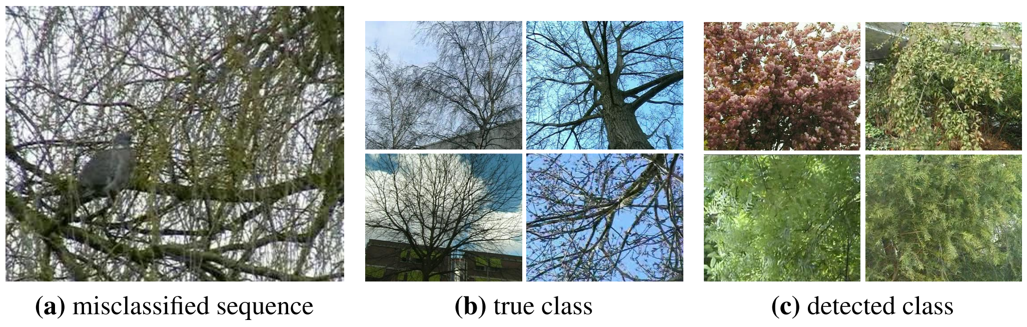
DynTex beta: The recognition rate on DynTex-beta is 99.38% with DT-AlexNet and 100% with DT-GoogleNet, which outperforms the state of the art s-TCoF [41] (99.38%). The only sequence misclassified with DT-AlexNet is illustrated in Figure 4.4a with some sequences of the true class (4.4b) and of the detected class (4.4c). This misclassification is due to the background of this sequence containing mainly a blue sky and trees and occupying the major spatial part of the sequence.

DynTex gamma: On the DynTex-gamma dataset, only one sequence is misclassified out of 264 by the DT-CNNs, improving the state of the art [41] from 98.11% to 99.62%. The 79th sequence of true class "naked trees" is misclassified as class



(a) misclassified sequence **(b)** true class **(c)** detected class

Figure 4.4: A misclassified sequence of the DynTex beta dataset and examples from the true class and from the detected one. (a) misclassified sequence, (b) true class "rotation" and (c) detected class "trees".



(a) misclassified sequence **(b)** true class **(c)** detected class

Figure 4.5: A misclassified sequence of the DynTex gamma dataset and examples from the true class and from the detected one. (a) misclassified sequence, (b) true class "naked trees" and (c) detected class "foliage".

"foliage" with a very similar appearance and dynamic. As depicted in Figure 4.5, the tree in the misclassified sequence is denser as compared to other trees in the same class "naked trees", which causes the confusion. This result is satisfying as DynTex-gamma contains relatively high intra-class variations (e.g. "fountains" and "flags") and inter-class similarities (e.g. "naked trees" vs. "foliage" and "sea" vs. "calm water") both in terms of spatial appearance and dynamics.

Dyntex++: In this experiment, the DT-GoogleNet (98.58%) also outperforms the best results in the literature [137] (98.48%). The classification rate of each of the 36 classes with DT-AlexNet (98.18%) is detailed in Figure 4.6. The most misclassifications occur for the classes with high intra-class variation "water fountain" and "smoke", with sequences which, with a closer look, do not always exhibit the expected DT due to the automatic splitting process of the original sequences from DynTex described in [39]. The confusion matrix is not illustrated due to the large number of classes, yet one may notice by visualising it that there are no dominant categories with which these misclassified sequences are confused (i.e. the confusions are spread over several classes). This experiment shows the effectiveness of the

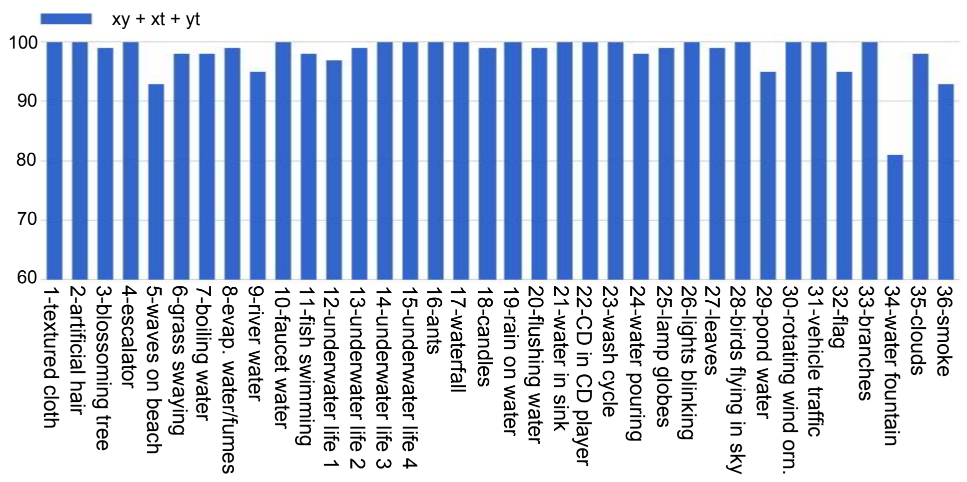


Figure 4.6: Classification rates of individual classes of the Dyntex++ dataset with the proposed DT-CNN approach (DT-AlexNet).

proposed approach with a relatively high number of training samples and classes as well as high intra-class variation.

The UCLA database contains only four sequences per primary category (50 classes setup). Therefore, the number of training sequences per primary category is three, which is much lower than the other datasets. CNNs require a high number of training samples to adjust the weights in order to learn meaningful features, generalise the recognition task to new unknown data, and avoid overfitting the training data. The database is not highly challenging because of low intra-class variation; hence, the proposed DT-CNNs are able to reach nearly 100% classification accuracy and are close to the state of the art. As a result of the training size, however, they do not outperform the best shallow hand-crafted methods in the literature on the three sub-datasets as described below.

UCLA-50: A classification rate of 99.5% is achieved with only one sequence misclassified out of 200 with both DT-CNNs. The best performance in the literature is 100%, achieved by DFS in [177]. However, DFS performs poorly with larger and more challenging datasets and is largely outperformed by the DT-CNN on Dyntex++ by over 8%.

UCLA-9: The proposed DT-AlexNet and DT-GoogleNet obtain 98.05% and 98.35% classification rate respectively, close to the best result 99.6% in the literature [152]. The confusion matrix obtained with DT-AlexNet is detailed in Table 4.4. The number of training samples of a class largely influences the recognition rate of the test samples of that same class. The “smoke” class contains only four samples and obtains the lowest 75% classification. These results confirm that the DT-CNNs perform best on large datasets with a high number of training samples per class.

UCLA-8: The proposed approach achieves a classification rate of 98.48% with DT-AlexNet and 99.02% with DT-GoogleNet, close to the state of the art on this

Table 4.4: Confusion matrix of the proposed DT-AlexNet on UCLA 9-class.

	boil	fire	flow.	pl.	fount	sea	sm.	wat.	wfall
boil	1	0	0	0	0	0	0	0	0
fire	0	0.97	0	0	0	0	0.03	0	0
flower	0	0	0.92	0	0.08	0	0	0	0
fount	0	0	0	0.96	0	0	0	0	0.04
plants	0	0	0	0	1	0	0	0	0
sea	0	0	0	0	0	0.97	0	0.03	0
smoke	0	0	0	0	0	0	0.75	0.25	0
water	0	0	0	0	0	0	0.03	0.97	0
wfalls	0	0	0	0.01	0	0	0	0	0.99

Table 4.5: Confusion matrix of the proposed DT-AlexNet on UCLA 8-class.

	boil	fire	flower	fount	sea	smoke	water	wfalls
boil	0.99	0.01	0	0	0	0	0	0
fire	0.01	0.99	0	0	0	0	0	0
flower	0	0	1	0	0	0	0	0
fount	0	0	0	1	0	0	0	0
sea	0	0	0	0	0.95	0.02	0.02	0.01
smoke	0	0	0	0	0	1	0	0
water	0	0	0	0	0	0	1	0
wfalls	0	0	0	0.01	0	0.02	0.01	0.96

Table 4.6: Accuracy results (%) of the proposed DT-AlexNet on multiple DT datasets using various combinations of planes.

	Dyntex++	DynTex-alpha	DynTex-beta	DynTex-gamma	UCLA-50	UCLA-9	UCLA-8
xy	94.28	**100**	98.77	98.11	99	95.7	98.04
xt	96.57	**100**	95.68	97.35	98.5	97.4	95.76
yt	96.28	**100**	95.06	97.73	93	97.15	95.65
$xy+xt$	97.57	**100**	**99.38**	99.24	**99.5**	97.4	98.26
$xy+yt$	97.71	**100**	**99.38**	**99.62**	99	97.8	**98.59**
$xt+yt$	97.84	**100**	97.53	98.11	98	97.95	96.85
$xy+xt+yt$	**98.18**	**100**	**99.38**	**99.62**	**99.5**	**98.05**	98.48

sub-dataset of 99.4% [152]. The confusion matrix in Table 4.5 shows that the few confusions with DT-AlexNet mainly occur for classes which exhibit high similarities in the spatial appearance and/or dynamics (e.g. “sea”, “smoke”, “water” and “waterfalls”).

4.4.2 Contribution of the planes

The results obtained with DT-AlexNet using different combinations of planes are reported in Table 4.6.

Single plane analysis

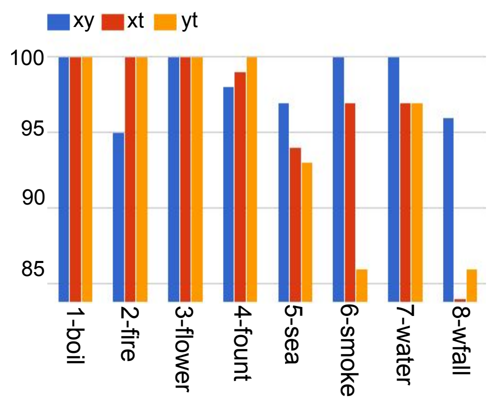
The spatial analysis of the *xy* plane performs better overall than the temporal ones (*xt* and *yt*). Yet, the temporal analyses are close to, and at times outperform the spatial analysis. Note that the performance of the *xt* and *yt* planes for certain classes may depend on the angle at which the DT videos are captured as important temporal information may be captured in one or the other plane. For instance, translational motions such as waves and traffic may be contained in only one of the temporal planes if the direction of the motion is parallel/perpendicular to one of the spatial axes. Furthermore, independent CNNs are finetuned on each plane and thus learn differently from the training data. Therefore, differences are noticeable between the temporal analyses of the *xt* and *yt* planes.

The good results obtained with both single temporal planes go against early conclusions made in the literature stating that DT recognition mostly relies on the spatial analysis while the motion analysis can only provide minor complementary information [39, 151, 179]. This statement is based on experimental results and on the human perception of DTs. It might be true for the experiments conducted in [39, 151, 179] and for a human being able to differentiate most DTs with a single frame. Yet, the proposed experiments demonstrate that it does not generalise to all the analysis methods, and that deep neural network approaches are excellent at finding their own ways to analyse data. Indeed, when it might be difficult for a human to recognise sequences of temporal slices like those illustrated in Figure 4.3b and 4.3c, the proposed approach is able to accurately classify sequences given only the temporal slices. Similar observations were made in [14] in which the LBP histograms on all three single planes result in similar accuracies on the DynTex dataset. Note that the analyses of the *xt* and *yt* planes are not purely temporal as they reflect the evolution of 1D spatial lines of pixels over time. Yet these planes exhibit temporal variations and this type of approaches is generally referred to as temporal analysis.

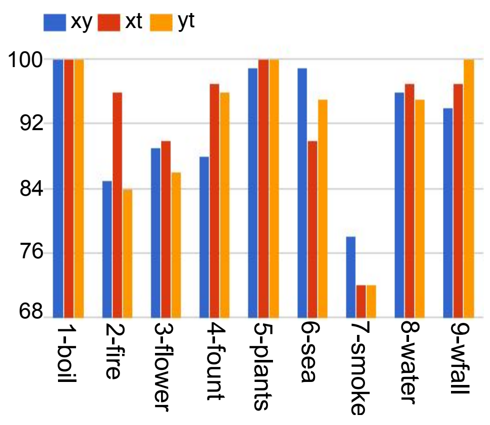
The spatial analysis performs 1% to 3% better than the temporal ones for most of the datasets (DynTex beta, DynTex gamma, UCLA-50 and UCLA-8), while the temporal analysis outperforms the spatial one on Dyntex++ and UCLA-9 by 2% and 1.45% respectively. The accuracy of each class using a single plane analysis is detailed in Figure 4.7a, 4.7b, 4.8a, 4.8b and 4.9 for the UCLA-8, UCLA-9, DynTex-beta, DynTex-gamma and Dyntex++ datasets respectively. No evident and consistent patterns emerge from the results across datasets to conclude and generalise what recognition the spatial analysis is able to achieve better than the temporal ones.

Combination of planes

As detailed in Table 4.6, the combination of all the planes results in a consistent increase of accuracy as compared to the sole spatial analysis. Similarly to [14], these results highlight the complementarity of the spatial and temporal planes. In [14], the authors heuristically assign weights to the contribution of the planes based on the recognition rate. Such experiments are not reported here in order to avoid heuristic

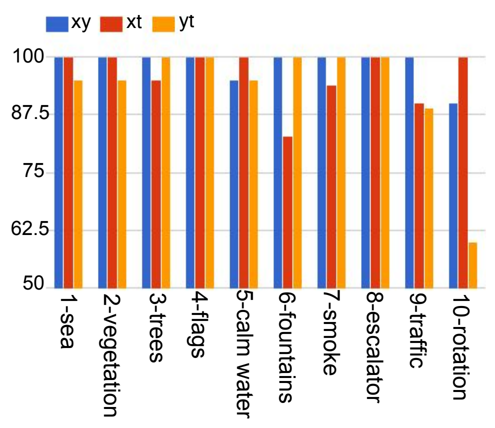


(a) ucla-8

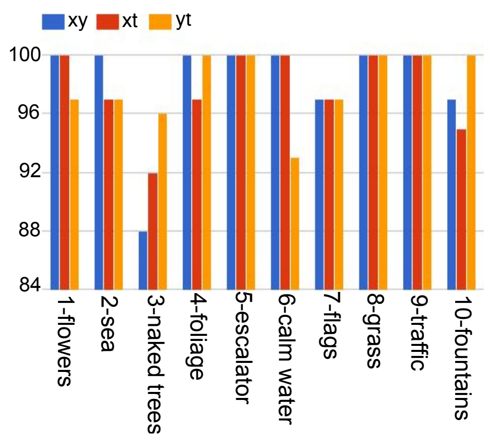


(b) ucla-9

Figure 4.7: Classification rates of individual classes using single *xy*, *xt*, and *yt* planes with DT-AlexNet on the (a) ucla-8 and (b) ucla-9 sub-datasets.



(a) DynTex beta



(b) DynTex gamma

Figure 4.8: Classification rates of individual classes using single *xy*, *xt*, and *yt* planes with DT-AlexNet on the (a) DynTex beta and (b) DynTex gamma sub-datasets.

and biased results. Also, learned parameters do not always generalise similarly to the test set; e.g. a low training loss can typically result in a low test accuracy if the network overfits the training data. For this reason, the weights are not assigned based on the training loss or any other measure of performance calculated on the training set.

The *xy* plane captures the spatial information while the other two planes mainly capture the dynamics of the sequences. As expected, combining the spatial plane to a single temporal plane performs generally better than the sole spatial plane or the combination of the two temporal planes. The best overall performance is achieved by the combination of the three planes which again highlights their complementarity.

Finally, note that for some datasets, the temporal slices *xt* and *yt* could be probabilistically expected to exhibit the same dynamic. It would be the case for

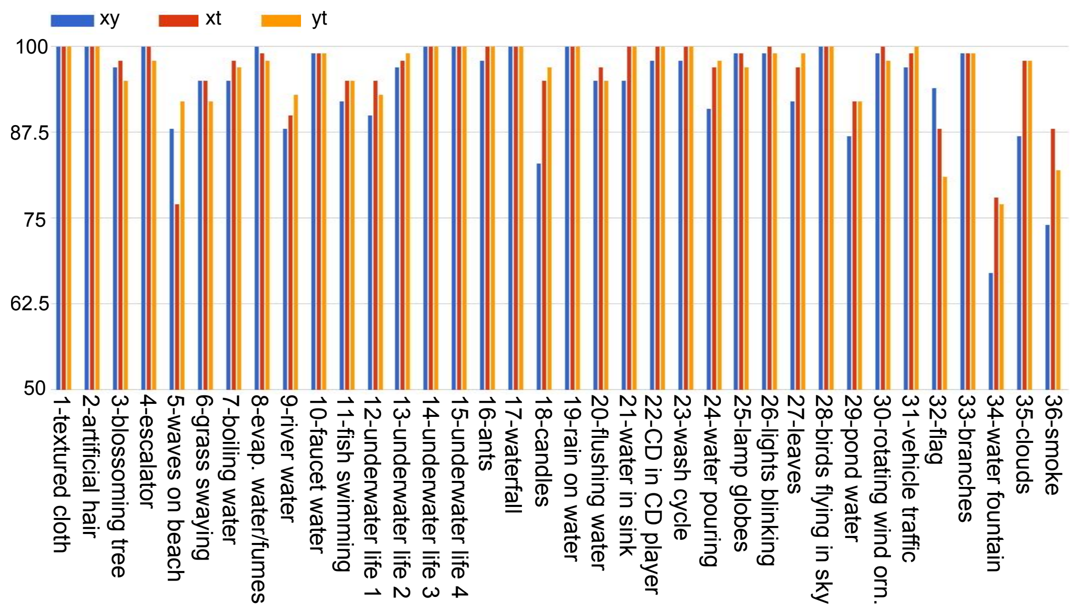


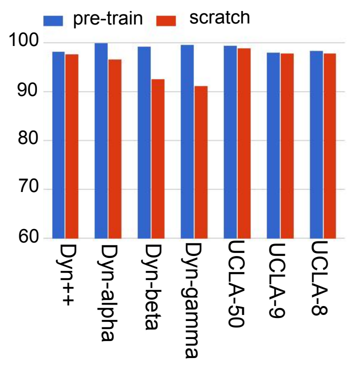
Figure 4.9: Classification rates of individual classes of the Dyntex++ dataset using single *xy*, *xt*, and *yt* planes with DT-AlexNet.

instance with random rotations of the field of view across different sequences or for DT sequences without dominant orientation of motion. In such scenario, it could be useful to combine the temporal slices *xt* and *yt* into a single analysis, i.e. using two finetuned networks instead of three and obtain more rotation invariance and more training data for the temporal plane. However, this approach has not resulted in an increase of accuracy in the experiments proposed in this chapter.

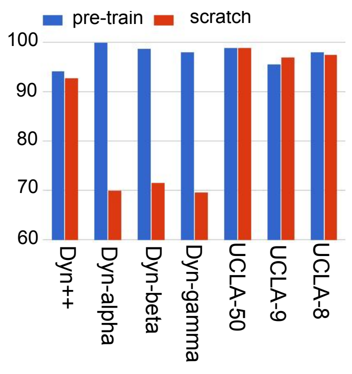
4.4.3 Domain transferability and visualisation

A comparison of DT-AlexNet using networks from scratch and pre-trained is depicted in Figure 4.10. Pre-training the networks only slightly improves the accuracy of the proposed method using the combination of three planes (see Figure 4.10a). The networks are able to learn from scratch due to the relatively large number of samples resulting from the slicing approach and to the low intra-class variation and high inter-class separability of some of the datasets. As shown in Figure 4.10, the smaller T-CNN3-S network used for Dyntex++ and UCLA learns better from scratch than the larger T-CNNs used for the DynTex datasets. This observation is particularly evident for single plane methods (Figure 4.10b to 4.10d). With fewer parameters, they learn better from small datasets with less overfitting and thus do not benefit as much from the pre-training.

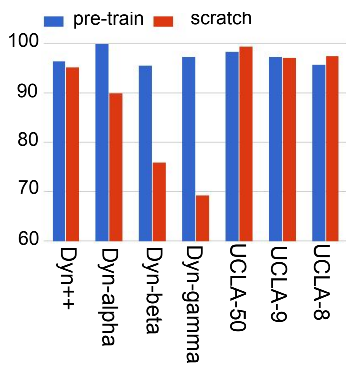
Moreover, one could expect the spatial analysis to benefit extensively more from the pre-training than the temporal analysis as the source (ImageNet) and target (spatial textures) domains are closer. Yet, the temporal planes benefit almost similarly,



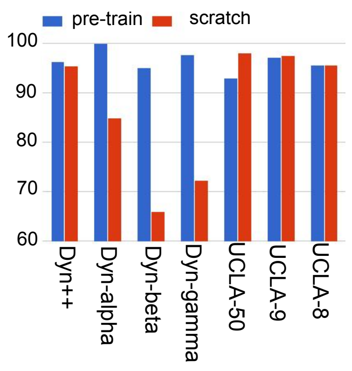
(a) xy+xt+yt



(b) xy



(c) xt



(d) yt

Figure 4.10: Classification rates of DT-AlexNet with networks trained from scratch vs. pre-trained on ImageNet with the following planes: (a) *xy+xt+yt* (b) xy, (c) xt and (d) yt.

which demonstrates the transferability of the learned parameters across domains. It also indicates that the learning is not biased by an overlap (i.e. similar images and classes) between the pre-training ImageNet dataset and the DT datasets.

Visualising the kernels learned by the networks and the features that they detect can provide an insight of what is learned during training as explained in Appendix A.4.7. Some classes in ImageNet contain images with high texture content such as “cheetah”, “chainlink fence”, “theater curtains” and “grille”. As demonstrated in Chapter 3, pre-training a T-CNN with such classes transfers better on texture datasets than pre-training with object-like classes. The networks learn repetitive texture patterns from these classes, which transfer well to spatial and temporal texture slices in DTs. Inputting spatial or temporal slices from the DynTex database to T-CNN pre-trained on ImageNet often highly activates neurons that learned to recognise these classes with high texture content, even though the patterns are not identical. This suggests that the repetitivity is learned by the networks to some extent independently from the patterns shapes. Finally, the response at the last convolution layer of the T-CNN to both spatial and temporal DT slices is denser than the response to an object image and is repetitive across the feature map. This is also expected and motivates the use of the energy layer for texture and DT analysis.

4.5 Discussion

A new DT-CNN approach has been designed for the analysis of DT sequences based on convolutional networks applied on three orthogonal planes. The T-CNN architecture proposed in the previous chapter has been applied to DT recognition and another architecture has been developed for smaller images (Dyntex++ and UCLA). Training independent networks on three orthogonal planes and combining their outputs in an ensemble model has obtained conclusive results on DT classification by learning to jointly recognise and pool spatial and dynamic features.

The approach has been evaluated on various datasets commonly used in the literature. The deepest architecture (DT-GoogleNet) obtained slightly higher accuracy than the shallower one (DT-AlexNet) and has established a new state of the art on the DynTex and Dyntex++ databases. The deep learning process enables the developed approach to analyse and learn from many diverse datasets and setups with a good invariance to rotation, illumination, scale, sequence length and camera motion. The developed DT-CNNs achieved high accuracy (>98%) on all the tested datasets; whereas previous methods in the literature are generally specialised in the analysis of one particular database and fail for other ones (see Table 4.3). The saturation of the results on these experiments has also revealed the lack of larger and more challenging DT datasets to fully exploit and evaluate deep learning methods.

High accuracies have been obtained using single temporal planes, at times outperforming the spatial analysis. It demonstrates both the domain transferability of the features learned only on spatial images and the importance of temporal analysis in DT recognition. Finally, the results have demonstrated the complementarity of the spatial and temporal analyses with the best results obtained by the combination of the three planes over single and two planes analyses.

Chapter 5

Texture segmentation with fully convolutional networks

5.1 Introduction

The previous chapters have presented conclusive results in the classification of texture and DT using convolutional networks. This chapter investigates the use of Fully Convolutional Networks (FCNs) [6] (see Appendix A.4.5) for the segmentation of texture images. A new architecture, referred to as FCNT (Fully Convolutional Network for Texture) is developed for texture segmentation. This network discards very deep shape information and combines the responses to filter banks at various depths to make use of local information from shallow features and more global and abstract information from deeper ones.

This chapter differentiates between two major texture segmentation problems with different application focuses. The first problem concerns the semantic segmentation of types of textures with a potentially large intra-class variation (e.g. wood, grass, and textile). This is relatively similar to a classic image classification or segmentation task in which an algorithm is trained to detect an object with many possible variations (shape, scale, lighting, etc.) in a supervised manner with many samples of each class. The second problem is to segment textures with a single training sample (supervised) or with no training data at all (unsupervised). Note that the approach is unsupervised but the training of FCN itself is supervised as explained in Section 5.2. For this second problem in particular, the repetitivity of the textural patterns is used which can be efficiently learned by finetuning a FCN with little training data. Three sets of experiments are conducted to evaluate the proposed approach including A) a supervised semantic segmentation with multiple training images, B) a supervised segmentation with a single training image per class and, C) an unsupervised segmentation. The developed approach is evaluated on images generated specifically for experiment A, as well as on the Prague texture segmentation benchmark [26] (experiments B and C) with a significant improvement of the state of the art.

The main contributions of this chapter include the following:
(1) A FCN architecture is developed for texture segmentation combining the analysis of simple local texture patterns with more global and complex patterns while discarding the overall shape analysis;
(2) it is demonstrated that such network can learn to segment textures from non-segmented training texture images;
(3) the developed FCNT can also be trained with very little training data (single image per class and even parts of the test image itself) which enables learning texture segmentation in an unsupervised framework;
(4) a segmentation refinement method is developed based on simple mathematical morphology and information contained in the network output generally discarded by classic inference.

The rest of this chapter is organised as follows: The developed approach is described in Section 5.2 including network architecture, training, and segmentation refinement methods. Three sets of experiments (A, B and C) are then presented in Section 5.3 including supervised and unsupervised texture segmentation. Finally, a discussion is provided in Section 5.4.

5.2 Material and Methods

This section describes the FCNT architecture for texture segmentation as well as a simple segmentation refinement method[1].

5.2.1 Network architecture

As demonstrated in the previous chapters, the overall shape analysis processed in the deepest layers can be discarded for the analysis of texture. Therefore, a fully convolutional architecture is developed based on the FCN8 network [6] with fewer convolution layers, based on the same idea as T-CNN (see Chapter 3).

Segmentation difficulties can occur, in particular across texture boundaries, due to downsampling in deep networks, resulting in large receptive fields with high abstraction and a loss of local information. This issue can be addressed by using skip connections [6] in FCNs in order to combine local and global information in the deconvolution layers. This is particularly relevant in texture segmentation as it does not only improve the boundaries segmentation but also provides high-frequency details and analysis of small simple texture patterns. Where the FCN8

[1]The network architecture implemented with Caffe [165] and more details are available on GitHub: https://github.com/v-andrearczyk/FCNT

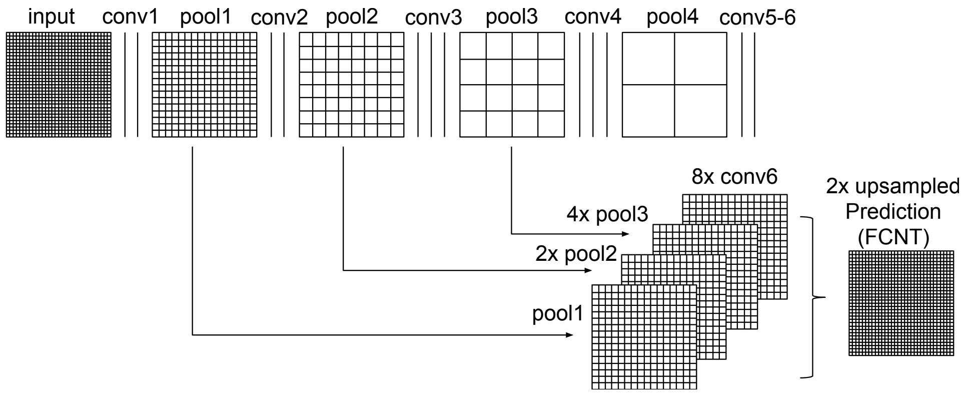


Figure 5.1: The FCNT architecture inspired from [6]. The grids reveal the relative spatial coarseness of pooling and prediction layers. Convolution layers are depicted as vertical lines. Skip connections, represented by arrows, allow the network to combine local information from early layers with more global information extracted by deeper layers (conv6). Diagram adapted from [6].

fuses the outputs of the third, fourth and seventh convolution blocks using skip layers, the developed network aggregates information of the first, second, third and sixth convolution blocks (conv1, conv2, conv3, and conv6) instead. The architecture of the proposed FCNT is illustrated in Figure 5.1. The upsampling is performed by successive deconvolution operations. Note that conv5 and conv6 are the equivalents of fully-connected layers in classic CNN architectures.

5.2.2 Refinement of segmented regions

As explained in the previous section, local information from shallow layers is used to significantly improve the boundary segmentation. Nevertheless, boundary pixels remain a difficulty for segmentation and a major cause of misclassification of the FCNT. For this reason, a segmentation refinement method is developed based on the output of the network. In the classic inference phase of a FCN, each pixel is assigned a prediction vector of size equal to the number of training classes. The segmentation result is obtained by assigning each pixel to the class which corresponds to the highest score in the corresponding prediction vector. However, more information is available as each class is assigned a score in these vectors. The idea of the refinement method is to make use of this extra information to improve the segmentation results and to obtain a single texture region per class in experiments B and C. To do so, *N* largest patches are isolated and filled (“largest patches” in Algorithm 1), where *N* is the number of texture classes in the segmented image to refine. A patch refers to a connected region that corresponds to an individual class label. As illustrated in Figure 5.2f, there is generally more than one patch per texture label before refinement. For instance, one large orange region detected the middle left wood texture, while multiple small other orange regions wrongly detected the same texture. In an iterative process, the small isolated regions (pixels not part of the largest patches) are then assigned the second best prediction of the network to attempt to merge them with larger patches. Successively, the remaining isolated regions are assigned to class labels with incrementally lower probability scores in the output vectors. The refinement stops when a segmentation with a single patch per class is obtained. The pseudo code for this approach is provided in Algorithm 1 below.

Algorithm 1 Pseudo code of the segmentation refinement method.

```
while $\bar{N} \neq N$ do
    for $n = 1..N$ do
        Relabel $n$, largest patches $\rightarrow im_n$
        if $im_{n-1} \neq im_n (n > 1)$ then
            break
        end if
    end for
end while
```

The number of patches and the number of texture classes are noted $\bar{N}$ and N respectively. "Largest patches" means that the largest patches of each class are filled, i.e. the labels of the small regions fully contained in the N largest patches are changed to the patches' labels. "Relabel n" refers to the relabelling of the pixels which are not part of the largest patches to the n^{th} best class prediction of the network at these pixel locations. Finally, im_n is the image output of "largest patches" at the current loop iteration. An example of refinement result is illustrated in Figure 5.2g (refined from Figure 5.2f).

5.3 Experiments

In this section, the developed method is evaluated on a set of supervised and unsupervised experiments. Three types of experiments (A, B and C) are conducted as follows:

5.3.1 Experiment A: Supervised training with multiple training images per class

This task is relatively similar to a classic FCN training method in the sense that multiple training instances of the same class are used. The intra-class variation forces the network to learn certain invariances to scale, rotation, and even types of textures. However, the major differences with classic object segmentation problems are the

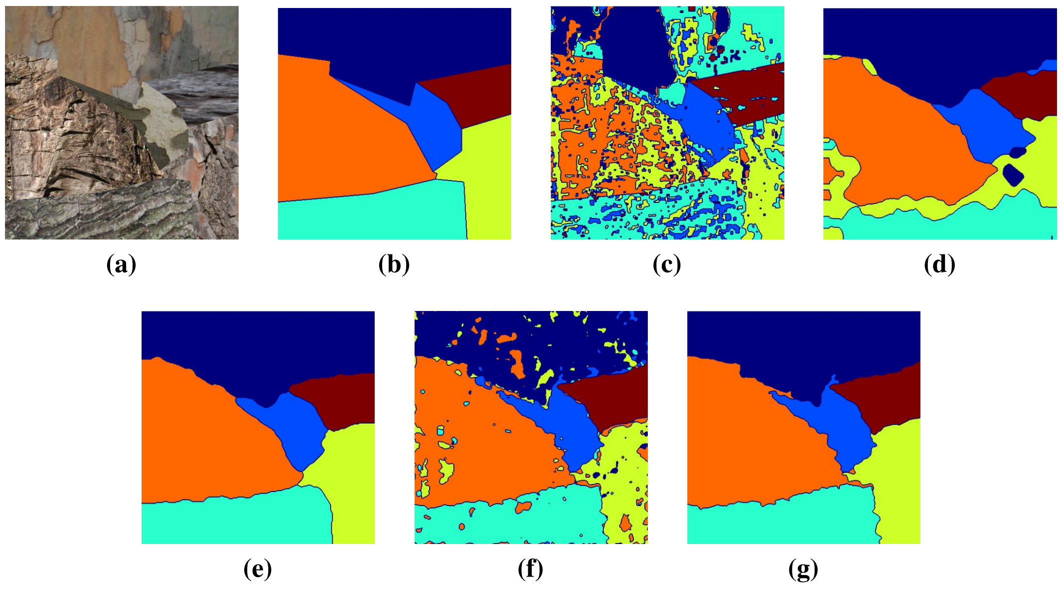


Figure 5.2: Examples of segmentation with FCNT on the supervised Prague texture segmentation task (experiment B) and comparison with the state of the art (see Section 5.3.2 for more details on the state of the art methods). The models are trained with one image per texture segment (six training images in this example). (a) Input image to segment; (b) ground truth segmentation; (c) Markov Random Field (MRF) segmentation [120]; (d) Co-Occurrence Features (COF) segmentation; (e) Con-Col segmentation; (f) FCNT segmentation without refinement; (g) FCNT segmentation with refinement. Best viewed in colour.

texture nature of the images and the fact that the training images are not segmented. This means that the texture properties are homogeneous across an entire training image and a single class label is given to all the pixels.

Datasets

For this experiment, two datasets are introduced as follows: For the **kth-seg** dataset, images are taken from the kth-tips-2b dataset [22] which contains 11 classes of texture images (see Section 3.3). Each class is made of four groups and each group contains images at nine different scales. Images at scale 10 are used, therefore each group in a particular class of the derived kth-seg dataset contains 12 images for a total of 528 images. The evaluation is based on the idea of 4-fold cross-validation used in [8, 25]. Each group is used once to create test images of multiple texture regions while the remaining groups are used for training. Two to five images are randomly picked from the test set to create a mosaic image with one texture region from each image (all from different classes). For each fold, there is a total of 396 training images and 80 test images. For the **Kylberg-seg** dataset, images are taken from the Kylberg texture classification dataset [31] which contains 28 classes of 160 images each. One of 12 available orientations is randomly selected for each image similarly to Section 3.3. Half of the dataset is used for training, the rest to create test images. The 576×576 images are split into four subimages of size 288×288. Images from the test set are randomly picked to create test images of size 288×288 with two to five texture regions from different classes, similarly to kth-seg. There is a total of 8,960 training images and 80 test images. Examples of the developed Kylberg-seg dataset are shown in the first row of Figure 5.3.

Note that in such supervised approach with non-segmented images, training mosaic images could be created to help the network in taking boundary decisions. However, this would be heavily biased, as the artificial segmentation would be very similar to the test segmentation. This approach might not generalise well to real life texture segmentation problems as the boundary between two real textures can be curved, melted, blurred, etc.

Details of the network

In this experiment, the developed FCNT network is compared with the original FCN8 [6]. Both networks are initialised with the weights learned with FCN8 on the PASCAL VOC 2011 dataset [180]. Layers which are not common to both networks (fully-connected and deconvolution layers) are initialised with the Xavier method [181] (see Appendix A.2.3). The width of the network in the deconvolution layers (number of feature maps in the last fully connected layer and following layers) is equal to the number of training texture classes, i.e. 11 for kth-seg and 28 for Kylberg-seg. The networks are trained with stochastic gradient descent similarly to [6]. The amount of training data is relatively large (as compared to experiments B and C), and the networks need to learn a high amount of information and variation. Therefore, the networks are trained for 300,000 and 400,000 iterations for the kth-seg and Kylberg-seg experiments respectively. Neither pre-processing, post-processing nor refinement is applied to demonstrate only the discriminative power of FCNs in texture segmentation with multiple training images per class.

Results

In this experiment, the FCN8 developed in [6] is compared with the developed FCNT network. The results are summarised in Table 5.1 as the average correct assignment of pixels, referred to as CO and explained in Section 2.2.3. Figure 5.3 illustrates examples of segmentation results with two to five texture regions per test image. Note that the segmentation results are directly given by the output of the network which learned to recognise multiple classes from non-segmented training images. The kth-seg and Kylberg-seg datasets contain 11 and 28 classes respectively with challenging inter-class similarities and a high intra-class variation for kth-seg. The

Table 5.1: Average correct pixel assignment (CO) of the proposed segmentation with FCN8 [6] and FCNT networks on the developed kth-seg and Kylberg-seg datasets (experiment A). The numbers next to the datasets (e.g. kth-X) represent the number of texture regions per test image.

	kth-2	kth-3	kth-4	kth-5	Kylb-2	Kylb-3	Kylb-4	Kylb-5
FCN8	67.93	72.60	67.76	**64.98**	85.94	79.13	77.43	76.33
FCNT	**68.70**	**73.05**	**68.31**	63.60	**88.81**	**80.59**	**79.11**	**76.80**

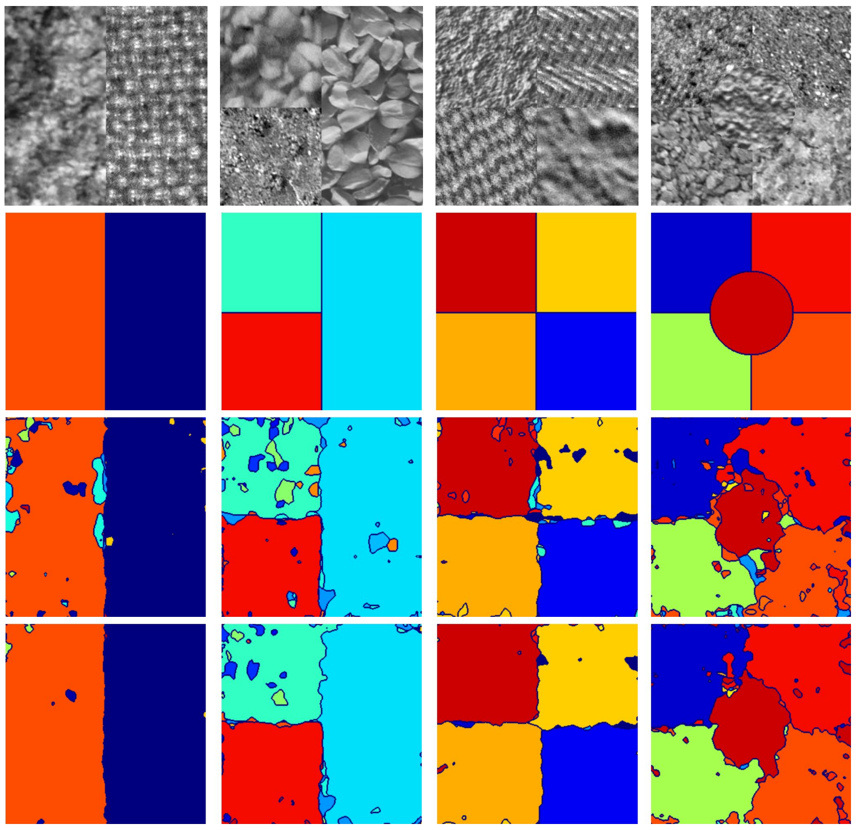


Figure 5.3: Examples of segmentation results with FCN8 [6] and with the developed FCNT on experiment A (Kylberg-seg). First row: input test image to segment; second row: ground truth segmentation; third row: FCN8 segmentation; fourth row: FCNT segmentation. Best viewed in colour.

FCNT outperforms FCN8 on the developed texture segmentation datasets (except for kth-5) which highlights the importance of using local information and discarding the overall shape analysis. The best architecture for a given application, however, might depend on the complexity of the texture patterns, the scale variation, the number of texture regions, as well as the desired results (e.g. precise boundary decision, minimum over-segmentation, etc.).

5.3.2 Experiment B: Supervised training with single training image per class

This experiment explores the application of the FCNT in a supervised segmentation task using one training image per class, i.e. per texture region.

Dataset

The Prague texture segmentation benchmark [26] facilitates the testing and comparison of algorithms on a range of supervised and unsupervised texture segmentation tasks. In this experiment, the supervised dataset with 20 test images is used. Each test image contains a number of texture regions of different classes ranging from 3 to 12 and a single training image is provided for each class, i.e. N training images for a test image containing N texture regions. Although different images are used for the training and testing sets, the variation between training and testing images is limited due to their visual similarity.

Details of the network

The FCNT is, like in experiment A, initialised with the weights learned by FCN8 on the PASCAL VOC 2011 dataset, and layers which are not common to both networks are initialised with the Xavier method. The width of the network in the deconvolution layers is again equal to the number of training texture classes which is given for each test image by the number of training images (3 to 12), and in turn is equal to the number of ground truth regions in the test image. The networks are trained for 5,000 iterations similarly to [6], although the results are stable to small changes of this number with a fast learning and negligible overfit.

Results

For this experiment, the developed method is compared to three algorithms whose results are provided on the Prague texture segmentation website [182]. The algorithm referred to as MRF is an implementation of the method developed in [120], an image segmentation algorithm using a Markov Random Field pixel classification model. Little information is provided about the second algorithm, referred to as COF as it uses co-occurrence features with a nearest neighbour classifier. Finally, no information is provided regarding the third method which reports the best results on this supervised dataset. This last algorithm is referred to as Con-Col as named in [182].

The results of this experiment are summarised in Table 5.2 in the form of various segmentation evaluations including region-based, pixel-wise and consistency mea-

Table 5.2: Results of experiment B on the Prague supervised dataset (normal size) and comparison with the state of the art. The results of the developed FCNT before segmentation refinement are referred to as FCNT-NR (no refinement). The performance measures are described in Section 2.2.3. Up arrows in the second column indicate that larger values correspond to better results and down arrows the opposite. Results marked with * indicate that no publication is currently known.

Measures		MRF*	COF*	Con-Col*	FCNT-NR	FCNT
Region-based	$\uparrow$ CS	46.11	52.48	84.57	87.52	**96.01**
	$\downarrow$ OS	0.81	**0.00**	**0.00**	**0.00**	1.56
	$\downarrow$ US	4.18	1.94	1.70	**0.00**	1.20
	$\downarrow$ ME	44.82	41.55	9.50	6.70	**0.78**
	$\downarrow$ NE	45.29	40.97	10.22	6.90	**0.89**
Pixel-wise	$\uparrow$ CA	65.42	67.01	86.21	87.08	**93.95**
	$\uparrow$ CO	76.19	77.86	92.02	92.61	**96.73**
	$\uparrow$ CC	80.30	78.34	92.68	93.26	**97.02**
	$\uparrow$ EA	75.40	76.21	91.72	92.68	**96.68**
	$\uparrow$ MS	64.29	66.79	88.03	88.92	**95.10**
	$\uparrow$ CI	76.69	77.05	92.02	92.81	**96.77**
	$\downarrow$ O	14.52	20.74	7.00	7.49	**2.72**
	$\downarrow$ C	16.77	22.10	5.34	6.16	**2.29**
	$\downarrow$ I.	23.81	22.14	7.98	7.39	**3.27**
	$\downarrow$ II.	4.82	4.40	1.70	1.49	**0.68**
	$\downarrow$ RM	6.43	4.47	2.08	1.38	**0.86**
Consistency	$\downarrow$ GCE	25.79	23.94	11.76	12.54	**5.55**
	$\downarrow$ LCE	20.68	19.69	8.61	9.94	**3.75**

sures (see subsection “Texture segmentation” in Section 2.2.3 for more information on these performance measures). The proposed FCNT largely outperforms the best results from the literature and significantly benefits from the segmentation refinement method (see Section 5.2.2). Figure 5.2 illustrates examples of segmentation results of the three algorithms from the literature and of the FCNT method with and without segmentation refinement.

5.3.3 Experiment C: Unsupervised training

In this experiment, the FCNT is evaluated on an unsupervised texture segmentation task, using patches extracted from the image as training data.

Dataset

In this last experiment, the Prague unsupervised texture segmentation dataset (ICPR 2014 contest [26]) is used. It contains 80 test images, including the 20 test images of experiment B, without any training sample. Each test image contains 3 to 12 texture regions which should be segmented as individual textures.

Pre-segmentation

While this experiment is unsupervised, labels are required to train the network and therefore a rough pre-segmentation map must be obtained from the test images. The idea is that the FCNT can learn from little training texture data due to the homogeneity and repetitivity across the regions of the test images. Therefore, the network can be trained from patches of texture regions roughly segmented in an unsupervised manner from the test images. Two methods are tested to obtain a pre-segmentation map.

First, a simple K-means clustering method is used. To that end, the input image to the pre-trained network is downsampled with bilinear interpolation by a factor of four in order to obtain more homogeneous label regions and a fast clustering. A K-means algorithm with a known number of clusters is run on the output of the FCNT. The K-means clustering is performed in a feature space of dimensionality equal to the number of classes in the pre-trained FCNT network. In this case, the dimensionality is 11 as it is pre-trained on kth-tips-2b. Note that in the ICPR 2014 contest [26], the number of clusters is unknown. This experiment is designed to demonstrate the power of the FCNT in a texture segmentation with rough pre-segmentation and not to evaluate such pre-segmentation methods. A known number of clusters K is therefore used as it provides a simple pre-segmentation map. The problem of obtaining more accurate pre-segmentation labels from the network with an unknown number of textures is kept for future work. The obtained label map is then upsampled by the same factor used to downsample the input image. The largest connected label region for each class is filled and used for training while other small isolated regions and dilated borders between regions are not used.

In a second approach, shallow segmentation algorithms from the literature are used to obtain the pre-segmentation labels. The four algorithms, with segmentation results provided in [182], are described in the following text. The Factorisation based texture Segmentation (FSEG) [118] uses the local distribution of filter responses (local spectral histograms, see Section 2.2) to construct a feature matrix. This matrix is factorised based on singular value decomposition into two matrices, one containing representative features, the other containing the weights used for linear combination of the representative features at each pixel location. These weights indicate the resulting segmentation map. The variational multi-phase segmentation framework (PCA-MS) developed in [116] also uses a filter bank approach. It performs a PCA dimensionality reduction of the extracted local spectral histograms and an energy (Mumford-Shah) minimisation segmentation. The Priority Multi-Class Flooding Algorithm (VRA-PMCFA) [183] involves a voting selection of a parameterised number of representative blocks of pixels based on wavelet distributions, followed by an iterative computation of segmentation maps by clustering, flooding and region merging. Finally, the last method achieves the state of the art on the unsupervised Prague dataset [182], although no information is provided about the algorithm. This method is referred to as MK in reference to the author's name (Martin Kiechle).

Details of the network

When using a K-means pre-segmentation, the FCNT is pre-trained on kth-tips-2b to obtain a first output without finetuning.

In order to train the network with the pre-segmentation map generated from K-means or from other algorithms, the FCNT is initialised similarly to experiments A and B. The test image itself is fed as training data with the pre-segmentation map as training labels. The width of the network in the deconvolution layers is equal to the number of training texture classes, given for each test image by the number of classes present in the pre-segmentation map.

As patches of the test image are used for training with rough pre-segmentation labels, the overfitting should be avoided. An assumption must be made that a majority of pixels in the pre-segmentation map are correctly classified and that the FCNT will first learn to segment these pixels which form a region of homogeneous texture properties. Misclassified pixels in the pre-segmentation map should exhibit a different texture. Yet, if it overfits, the network will learn a representation that merges these outliers and the correct regions into the same class. This is due to both the plasticity of the network and the small amount of training data. In order to avoid overfitting, the output of the network is evaluated on the test image during training. The training is stopped P iterations after all the texture classes present in the pre-segmentation map are detected in the output of the trained network. In other words, when at least one pixel is assigned to each class in the output segmentation map, the network is only allowed to learn for P more iterations. This early stopping method avoids overfitting the pre-segmentation training labels. Note that the number of iterations is limited to Q if the previous condition is not reached. This limit avoids overfitting an over-segmented training map in certain scenarios. In Figure 5.4d, for instance, the orange over-segmented region present in Figure 5.4c has been discarded, whereas the red one has been maintained. The parameters P and Q have been set by trial and error to 60 and 400 respectively.

Results

The results are reported in Table 5.3 with the same evaluation measures as in experiment B. Images of segmentation results obtained with the shallow algorithms as well as the FCNT approach with different pre-segmentation methods are illustrated in Figure 5.4. The FCNT used with existing pre-segmentation labels (FSEG, PCA-MS, VRA-PMCFA and MK) consistently improves the results from the segmentation of

Table 5.3: Results of the FCNT approach with various pre-segmentation methods on the Prague unsupervised dataset (large size) and comparison with the state of the art. Results of FSEG, PCA-MS, VRA-PMCFA and MK are reported as given on the Prague texture dataset website [182]. Up arrows in the second column indicate that larger values correspond to better results and down arrows indicate the opposite. Results marked with * indicate that no publication is known at the time of writing.

	Method	FCNT	FSEG	FCNT	PCA-MS	FCNT	VRA-PMCFA*	FCNT	MK*	FCNT
	labels	K-means	-	FSEG	-	PCA-MS	-	VRA-PMCFA	-	MK
Region-based	↑ CS	62.59	69.24	71.90	72.27	75.00	75.14	75.62	77.73	**79.34**
	↓ OS	13.02	12.28	**9.82**	18.33	17.09	12.13	11.51	15.92	13.67
	↓ US	16.31	17.03	19.86	9.41	9.69	9.85	10.17	6.31	**6.25**
	↓ ME	13.98	7.71	4.73	4.19	**3.62**	4.38	4.76	3.93	3.80
	↓ NE	14.09	6.89	4.26	3.92	**3.32**	4.37	4.66	3.92	3.80
Pixel-wise	↑ CA	74.58	76.32	78.67	81.13	83.02	83.45	83.77	82.80	**84.17**
	↑ CO	82.79	84.05	85.62	85.96	87.41	88.12	**88.54**	86.89	87.97
	↑ CC	83.90	84.15	84.35	91.24	91.77	90.73	90.51	93.65	**94.15**
	↑ EA	81.42	82.53	83.71	87.08	88.20	88.07	88.23	88.03	**88.97**
	↑ MS	74.18	77.11	79.18	81.84	83.61	83.92	84.28	83.98	**85.23**
	↑ CI	82.30	83.26	84.32	87.81	88.87	88.72	88.80	89.03	**89.91**
	↓ O	10.43	11.66	9.99	7.25	5.34	**4.51**	4.76	7.68	6.47
	↓ C	12.45	11.71	8.46	6.44	**5.53**	8.89	7.81	24.24	22.88
	↓ I.	17.21	15.95	14.38	14.04	12.59	11.88	**11.46**	13.11	12.03
	↓ II.	3.56	3.29	3.11	1.59	1.47	1.48	1.48	1.50	**1.42**
	↓ RM	5.21	5.00	5.11	4.45	4.25	3.75	3.71	3.27	**3.12**
Consistency	↓ GCE	14.21	10.75	7.62	8.33	6.75	6.55	**6.39**	7.40	6.46
	↓ LCE	8.17	7.52	4.97	5.61	4.10	**3.90**	3.97	5.62	4.75

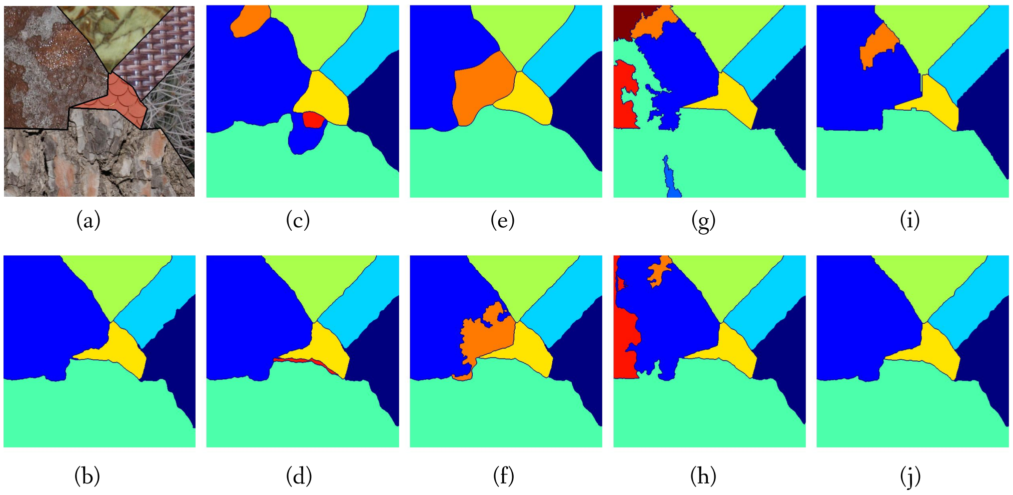


Figure 5.4: Examples of segmentation results using different methods on the Prague unsupervised dataset. (a) Input image with superimposed ground truth boundaries; (b) FCNT segmentation with K-means training; (c) FSEG segmentation; (d) FCNT segmentation with FSEG pre-segmentation; (e) PCA-MS segmentation; (f) FCNT segmentation with PCA-MS pre-segmentation; (f) PMCFA segmentation; (g) FCNT segmentation with PMCFA pre-segmentation; (h) MK segmentation; (i) FCNT segmentation with MK pre-segmentation; Best viewed in colour.

these shallow algorithms. The FCNT trained with labels from the MK segmentation largely outperforms the state of the art (MK segmentation). When compared to the literature, it obtains the best score on 12 out of 18 measurements and second best score on four others.

Note that these results are obtained on the Prague dataset referred to as “large” in [26] (80 test images), whereas the supervised results are presented in Table 5.2 for the “normal” Prague dataset (20 test images) in order to compare to the state of the art. A comparison between supervised and unsupervised approaches is possible as the results on the normal and large supervised datasets are very similar and the large dataset is an extended version of the normal one. The supervised FCNT method obtains 96.01% correct segmentation (CS) on the normal dataset and 95.64% on the large dataset; as expected, significantly higher than the unsupervised best result of 79.34%.

5.4 Discussion

This chapter has demonstrated the power of FCNs in the segmentation of texture regions. A fully convolutional architecture has been designed by combining shallow features encoding local high-frequency information and deeper features containing more abstract and lower frequency information. The developed FCNT can learn to perform semantic segmentation from classic texture recognition datasets with non-segmented textures such as kth-tips-2b and Kylberg datasets. This network can also be trained with a very small amount of training data due to the repetitivity and low-complexity of texture patterns. Indeed, the complexity of the texture patterns is limited as compared to objects or scenes, and these simple patterns are easier to learn via gradient descent. Numerous patterns at multiple scales and frequencies are generally present in a texture region and the FCNT is trained to recognise each pattern by learning filters at multiple scales, with a training label for every input pixel. Each pattern can be thought of as an individual training sample with certain variation from the others. These observations have been made in experiment B by using a single training image per class and in experiment C by using texture regions of the test image itself as training data. The FCNT has been tested on several tasks including challenging supervised and unsupervised segmentations. The proposed approach has largely improved the state of the art on the supervised and unsupervised Prague texture segmentation benchmarks.

Chapter 6

Conclusions and future work

This chapter summarises the thesis, highlights the contributions and limitations of the research that was carried out and suggests several directions of research for future work. In particular, Section 6.1 summarises the thesis and details the major and minor contributions. Conclusions are drawn from the challenges, results, and analyses developed throughout the dissertation. A discussion is proposed about the limitations of the developed methods and, more globally, of deep learning. Suggestions for future work are provided in Section 6.2 based on the limitations, possible improvements, and alternative research paths.

6.1 Contributions and conclusions

This thesis has examined the analysis of textures and DTs using convolutional networks and deep learning. Texture and DT analysis is crucial in various applications including remote sensing, industrial inspection, content-based image retrieval and biomedical imaging. The scope of the research has been limited to the classification of static and temporal textures as well as the segmentation of texture regions. The motivation has raised from the following observations. Firstly, classic shallow texture analysis methods often lack invariances to rotation, scale, noise, blur, etc. and of abstraction of concepts. In turn, these methods lack generalisation to unknown data in the context of high intra-class variation commonly encountered in texture analysis problems. Moreover, classic approaches also require designing hand-crafted features for particular applications, which may not perform well on other applications. A thorough literature review of texture and DT analysis has been presented in Chapter 2. Classic feature extraction methods have been introduced including statistical, model-based, filter banks, wavelets, LBP-like, as well as dictionary learning methods which are a transition between hand-crafted descriptors and deep neural networks. Secondly, CNNs and deep learning are currently the state of the art in a variety of machine learning tasks across fields including computer vision, natural language processing, and speech recognition. CNNs learn multiple levels of visual features from large amounts of training data in a hierarchy of concepts which performs very well at recognising, detecting, segmenting or generating complex objects and scenes. An introduction to neural networks, deep learning, and convolutional networks is provided in Appendix A and recent deep learning methods applied to texture and DT analysis have been presented in Chapter 2. A CNN can often be naively applied to solve most computer vision problems due to its powerful learning scheme, its good generalisation, and its flexibility. However, CNNs offer characteristics that are particularly well suited for textures, with an extraction of features of increasing complexity and increasing spatial supports throughout the network similar to a multiscale non-linear filter bank. While this filter bank idea is shared with numerous classic texture analysis methods, CNNs offer a powerful end-to-end training of the filters and of the classification or segmentation of filter responses. Thirdly, despite these similarities, CNNs are designed for object recognition and thus analyse the overall shape and layout of images, whereas textures would benefit from an orderless pooling and classification of relatively small and simple texture patterns. The learning of complex and large features is conducted in the deepest convolution and fully-connected layers of CNNs. These observations motivated the proposed adaptations of deep convolutional networks to the analysis of texture images and temporal textures.

The homogeneity and pattern repetitivity across texture images or regions, and across time for DTs, have been largely exploited in this research. It enables the orderless average pooling of feature maps activations for classification, similarly to an energy response in a classic filter bank method. It also allows patches of large images to be considered as individual samples of identical texture. Similarly, slices of a DT sequence sampled across a spatial or a temporal plane can be treated as individual samples providing variations of the same spatial or temporal texture. Moreover, it enables the training of a texture segmentation network from non-segmented training images, i.e. with a single homogeneous texture across the training image. Thus, ground truth segmentation labels defined by hand are not required as opposed to object segmentation datasets. Finally, it allows a network to be trained to segment texture regions in an image directly from patches of this same image.

6.1.1 List of contributions

The first set of major contributions has been presented in Chapter 3. A CNN architecture, referred to as T-CNN, has been developed for the classification of texture images. The overall shape and layout analysis of classic CNNs (AlexNet and GoogleNet) has been discarded by orderless average pooling of intermediate feature maps. This energy approach significantly reduces the computational complexity and memory consumption of the networks, while improving its performance. One limitation, however, is that the developed deep network (GoogleNet adapted to texture) does not reach the same accuracy as the FV encoding of deep convolution layers in [8]. Texture and shape analyses have also been combined to demonstrate their complementarity within an alternative new network architecture. An application of the developed T-CNN has been introduced for the particular problem of biomedical tissue images classification. The number of annotated biomedical images is often limited due to privacy and to the lack of experts or of diagnosed patients. However, the size of tissue images is large and, as mentioned previously, the textures are highly repetitive. A framework has therefore been introduced in which the images are split into subimages in order to increase the number of training samples and combine multiple predictions at test time in an ensemble approach.

The second major contribution has been introduced in Chapter 4 as the extension of the developed texture specific CNN to the spatiotemporal domain for the recognition of temporal textures. A framework has been designed which trains independent networks to recognise spatial and temporal slices in three orthogonal planes (*xy*, *xt*, and *yt*), regularly sampled along the temporal, horizontal, or vertical axes. Three independent prediction outputs are then summed in an ensemble model to recognise unknown DT sequences. Despite the lack of challenging datasets, the results have highlighted the complementarity of the spatial and temporal analyses, outperforming the state of the art on multiple benchmarks. The analysis of single and pairs of planes has revealed that the temporal slices carry highly discriminative information. Even though some temporal slices are relatively meaningless for humans (e.g. foliage in the *xt* and *yt* planes illustrated in Figure 4.3), the developed networks are able to extract informative and discriminative features from it, resulting in recognition rates close to the spatial analysis results (*xy* plane).

The third major contribution has been presented in Chapter 5 as the segmentation of texture regions using a new framework with an adapted fully convolutional architecture. FCNs classify every pixel in the input image into a given class. Some pixels can be misclassified, in particular across region boundaries since the network makes a prediction, for a given pixel, based on its neighbourhood. The neighbourhood of a boundary pixel may be part of different texture regions which may result in a wrong segmentation. This problem is mitigated in the segmentation of objects as the contour or shape in the large neighbourhood, together with training ground truth segmentation, provides information on the class of a given pixel. Textures, however, are characterised by statistical properties and simple repetitive patterns rather than their region shape and contour and must be analysed differently. A new architecture, named FCNT, has therefore been designed to discard the overall shape analysis and to combine filter responses at multiple scales. The local (“where”) information of early layers has been combined with the deeper and more global ("what") information. A segmentation refinement method has also been introduced based on the fact that a FCN outputs a probability prediction for every class it is trying to detect. Whereas only the highest prediction score is generally used as the segmentation result for a given pixel, simple mathematical morphology methods have been used with the full prediction vectors to grow and refine texture regions. This refinement method results in the merging of isolated misclassified pixels with larger regions and in a single texture region per class. It significantly improves the segmentation across boundaries and largely outperforms the state of the art in multiple supervised and unsupervised tasks. The experiments have demonstrated that the developed FCNT can learn to segment textures from images of a single homogeneous texture, i.e. without ground truth segmentation. A first set of experiments has demonstrated that it can semantically segment textures with high intra-class variation from relatively large texture classification datasets (kth-tips-2b and Kylberg). The second and third sets of experiments have demonstrated that little texture data is required to train the networks due to the repetitivity and low-complexity of texture patterns. Thus, a single training image per class is sufficient to finetune a pre-trained network. Alternatively, a patch of an unknown test image can also be used to train a network to segment that same image in an unsupervised scheme.

The domain transferability of pre-trained parameters has played an important role in each chapter, in particular for the challenging and small datasets in Chapters 3 and 5. Various analyses of the learning process and of the domain transferability have been conducted, including a depth analysis of the T-CNN, an evaluation of domain transferability of pre-trained parameters and the visualisation of patterns that neurons in trained networks respond to. Performance comparisons have also been conducted between the developed texture specific deep approaches, classic CNNs and shallow machine learning methods, as well as between supervised and unsupervised segmentation problems. These analyses have shown that networks pre-trained on a texture dataset transfer better to another texture analysis task than those pre-trained on an object recognition dataset. Yet, the size of the training set is predominantly important, which suggests that the proposed DT recognition method could benefit from a large texture classification dataset with millions of images. The visualisation methods explained in Appendix A.4.7 and used in Sections 3.4.4 and 4.4.3 have revealed that neurons in deep layers of the developed texture specific CNNs respond to simple repeated patterns when trained on texture images, even when the network was pre-trained on ImageNet. This visualisation has also suggested that the repetitivity of texture patterns is learned as an important concept, to some extent independently from the patterns shapes.

Besides these major contributions, minor contributions of this research are listed in the following paragraphs.

A preliminary work has been presented in [151] for DT recognition based on a hand-crafted statistical analysis of motion in the spatiotemporal domain and its combination with binary pattern descriptors. The proposed approach was derived from co-occurrence matrices [10, 60] to jointly analyse pairs of magnitudes and orientations of the optical flow extracted between consecutive frames of DT sequences. This work has not been included in the thesis as it significantly deviates from the proposed deep learning ideas and approaches.

Three subsets of ImageNet have been introduced in [25] with (i) high texture content images, (ii) object images and (iii) randomly selected classes. Each subset contains 28 classes out of the 1,000 available classes in ImageNet and includes all the training and validation images of the selected classes. These datasets can, for instance, be used to compare the effect of pre-training a texture classification network on different types of datasets as proposed in Section 3.3.

Two texture segmentation datasets have been developed in Section 5.3.1 based on existing texture classification databases. They have been designed to evaluate supervised texture segmentation methods trained with multiple images per class, each image exhibiting a single homogeneous texture. These texture segmentation datasets are available upon request for other researchers.

Algorithms have been made available at the following GitHub repositories. An implementation of the T-CNN with 3 convolution layers finetuned and trained from scratch on the kth-tips-2b database is available at: https://github.com/v-andrearczyk/caffe-TCNN. An implementation of the DT-CNN approach developed in Chapter 4 for the classification of DTs (Dyntex++) is provided at: https://github.com/v-andrearczyk/DT-CNN. Finally, an implementation of the FCNT and the segmentation refinement method developed in Chapter 5 for texture segmentation is available at: https://github.com/v-andrearczyk/FCNT.

In conclusion, this thesis has highlighted through various experiments and analyses that CNNs are, subject to simple adaptations, well suited for the analysis of static and temporal textures. This work has demonstrated that the orderless pooling of intermediate feature maps is more adapted to the analysis of textures than the overall shape and layout analysis of classic CNNs.

6.1.2 Limitations of deep learning

Deep learning and CNNs are the most popular machine learning methods and have achieved the state of the art in most applications. Yet, it may not necessarily always be the best solution to a machine learning problem. Several elements and limitations are suggested in this section to emphasise this point.

One drawback of deep neural networks is its memory requirement and computational complexity as compared to shallow machine learning algorithms. Although this is a system problem rapidly vanishing due to technological advances, a less complex model should always be preferred given equal performance [184]. Deep networks should, therefore, be chosen for their better performance rather than for the simplicity to apply them to various applications with little modification.

It has also been shown that CNNs are easily fooled. Trained CNNs, for instance, can achieve accurate recognition on a very large and challenging dataset; yet, an image can be modified in a way imperceptible to the human eye, which leads to a misclassification [185]. In [186], the authors demonstrated that generated adversarial images unrecognisable to humans are classified with high confidence by neural networks as one of the training classes. Although adversarial images can also be generated for classic machine learning algorithms, a certain knowledge about what a system should learn can be directly incorporated into the design of hand-crafted features to avoid likely fooling occurrences.

Not integrating any knowledge about the domain and data into a system, and letting a CNN learn from images can be a source of erroneous model behaviour. There are scenarios in which a simple machine learning approach may be a better solution because an understanding of the problem and data can be used to engineer relevant hand-crafted features. For instance, if the training set consists of a small number of samples with low intra-class variation, a network may overfit. It may learn to recognise a feature which does not represent the class of interest, e.g. the colour of the background instead of an object of interest, and therefore will not generalise well to unknown data. This difficulty may also occur with the texture and DT analysis methods proposed in this thesis. With small training sets, it might be difficult to control what CNNs learn as discriminative features. It could be fooled, for instance, by orientation, illumination, scale, or artefacts and shapes which are only present in the training set as the invariance to these transformations must be learned through training. An analogy to the colour background example can be made in texture analysis. A network may learn to recognise a particular shape or object which is part of the training texture images rather than the texture of interest (e.g. a person on the beach, a bird in the sky, a brand logo on clothes or the contour of a shape containing a texture). An unknown image without this particular object or contour will be misclassified. Note that this problem is mitigated by the use of intermediate responses and the energy layer in the proposed TCNN approach, but many other examples can be found with non-representative training sets. On the other hand, invariance to specific variations, or knowledge of which type of feature should be sought can be voluntarily enforced in shallow classifiers.

These difficulties mainly concern small training datasets. As demonstrated in Chapter 4, the proposed TCNN was outperformed by classic shallow DT recognition algorithms on the smallest dataset while achieving significantly higher accuracy on large datasets. Note that methods can be employed to attempt to overcome these difficulties such as pre-training, synthetically increased training sets, learning to recognise adversarial examples, regularisation methods, domain adaptation, and Bayesian priors. However, designing appropriate simple hand-crafted features is easier and may perform better in some scenarios. CNNs might also be combined with hand-crafted knowledge and features in an ensemble manner.

A last important drawback of deep neural networks and of the solutions proposed in this thesis is not knowing what the networks learn and how they make a prediction. This problem, often referred to as the "black box", is related to the difficulty in tracing the prediction of a network back to important features, and revealing the internal process of a model. The interpretability of a model may be crucial in certain applications, for instance in medical image analysis, to obtain the approval of a diagnosis system or to find hidden structures and proceed with further analysis. Visualisation methods exist, however, to gain an insight into the complex functions, decisions, and features in CNNs. These methods were described in Appendix A.4.7 and used in Chapters 3 and 4. Finally, Bayesian modelling [111] can be used to understand what a network learns and why it makes a certain prediction for interpretability.

6.2 Future Work

Future research can be considered to overcome the limitations of the proposed approaches, to explore other research paths and to continue to advance the field of texture analysis using deep learning.

Firstly, large and challenging datasets should be developed to fully exploit and evaluate deep convolutional architectures in DT analysis. A solution to the difficulty of capturing a large database of videos with high variations and a large number of classes, could be to extract DT videos from the web. This approach, often referred to as "in the wild" as opposed to data obtained under controlled settings, is commonly used in computer vision. A texture recognition dataset is, for instance, developed in [99] with images obtained "in the wild".

Secondly, an interesting line of research for the recognition of DTs is to incorporate the analysis of the three planes into a single network architecture. A slow fusion could be used instead of the late fusion adopted in Chapter 4, by allowing connections in intermediate layers between features extracted from different planes. This approach would allow a network to learn how to combine the spatial and temporal information instead of learning it independently via multiple networks.

Thirdly, the fully convolutional method developed in Chapter 5 requires a pre-segmentation step for the unsupervised segmentation. The results highly depend on the obtained pre-segmentation map, in particular on the number of detected regions since an under- pre-segmentation cannot be corrected by the FCNT approach. Note that “under-segmentation” refers to the number of detected texture regions being lower than the ground truth number. Since the FCNT is trained to recognise the pre-segmented regions, it cannot detect more regions than these training instances. On the other hand, an over- pre-segmentation can be corrected by the FCNT approach, yet it is not always the case as illustrated in Figure 5.4. The best results have been obtained in the experiments by using shallow segmentation algorithms from the literature as label maps for training the networks. It has also been shown that the output of the network can be used for pre-segmentation using a K-means clustering method. This approach could be explored in more detail using robust clustering methods in the output feature space with spatial constraints.

Finally, the proposed methods have been successfully evaluated on various texture, biomedical tissue, and DT datasets. They could now be applied to various texture problems and, in particular, in biomedical imaging for the segmentation and recognition of lesions, nodules, and cancers or for the analysis of temporal textures exhibited in sequences of ultrasound or X-rays. Note that the T-CNN has been satisfyingly used by other authors, for instance in conjunction with haptic information for surface material classification [187] or in combination with other deep networks for image emotion recognition [188].

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[2] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[3] T. Leung and J. Malik, "Representing and recognizing the visual appearance of materials using three-dimensional textons," *International Journal of Computer Vision*, vol. 43, no. 1, pp. 29–44, 2001.

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

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[7] F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8, IEEE, 2007.

[8] M. Cimpoi, S. Maji, I. Kokkinos, and A. Vedaldi, "Deep filter banks for texture recognition, description, and segmentation," *International Journal of Computer Vision*, vol. 118, no. 1, pp. 65–94, 2016.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[10] R. M. Haralick, K. Shanmugam, *et al.*, "Textural features for image classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, no. 6, pp. 610–621, 1973.

[11] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *European Conference on Computer Vision*, pp. 143–156, Springer, 2010.

[12] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[13] T. Ojala, M. Pietikäinen, and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[14] G. Zhao and M. Pietikäinen, “Dynamic texture recognition using local binary patterns with an application to facial expressions,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, 2007.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[16] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

[17] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer Vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[18] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, “Return of the devil in the details: Delving deep into convolutional nets,” *arXiv preprint arXiv:1405.3531*, 2014.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[20] H. Jégou, M. Douze, C. Schmid, and P. Pérez, “Aggregating local descriptors into a compact image representation,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 3304–3311, IEEE, 2010.

[21] A. K. Jain and F. Farrokhnia, “Unsupervised texture segmentation using gabor filters,” *Pattern Recognition*, vol. 24, no. 12, pp. 1167–1186, 1991.

[22] E. Hayman, B. Caputo, M. Fritz, and J.-O. Eklundh, “On the significance of real-world conditions for material classification,” in *Computer Vision-ECCV 2004*, pp. 253–266, Springer, 2004.

[23] R. Polana and R. Nelson, “Temporal texture and activity recognition,” in *Motion-based recognition*, pp. 87–124, Springer, 1997.

[24] R. Péteri, S. Fazekas, and M. J. Huiskes, “DynTex: A comprehensive database of dynamic textures,” *Pattern Recognition Letters*, vol. 31, no. 12, pp. 1627–1632, 2010.

[25] V. Andrearczyk and P. F. Whelan, “Using filter banks in convolutional neural networks for texture classification,” *Pattern Recognition Letters*, vol. 84, pp. 63–69, 2016.

[26] M. Haindl and S. Mikes, “Texture segmentation benchmark,” in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, IEEE, 2008.

[27] G. R. Cross and A. K. Jain, “Markov random field texture models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 1, pp. 25–39, 1983.

[28] J. Portilla and E. P. Simoncelli, “A parametric texture model based on joint statistics of complex wavelet coefficients,” *International Journal of Computer Vision*, vol. 40, no. 1, pp. 49–70, 2000.

[29] J. J. Gibson, “The perception of the visual world.,” 1950.

[30] J. Malik and R. Rosenholtz, “Computing local surface orientation and shape from texture for curved surfaces,” *International Journal of Computer Vision*, vol. 23, no. 2, pp. 149–168, 1997.

[31] G. Kylberg, “The Kylberg Texture Dataset v. 1.0,” External report (Blue series) 35, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden, September 2011.

[32] M. Tuceryan, A. K. Jain, *et al.*, “Texture analysis,” *Handbook of Pattern Recognition and Computer Vision*, vol. 2, pp. 235–276, 1993.

[33] J. Chen and A. K. Jain, “A structural approach to identify defects in textured images,” in *Systems, Man, and Cybernetics, 1988. Proceedings of the 1988 IEEE International Conference on*, vol. 1, pp. 29–32, IEEE, 1988.

[34] F. Liu and R. W. Picard, “Periodicity, directionality, and randomness: Wold features for image modeling and retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 722–733, 1996.

[35] A. Depeursinge, O. S. Al-Kadi, and J. Mitchell, “Biomedical texture analysis,” *Fundamentals, Tools and Challenges (Academic Press, London, 2017)*, 2017.

[36] T.-Y. Lin and S. Maji, “Visualizing and understanding deep texture representations,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2791–2799, 2016.

[37] N. Jetchev, U. Bergmann, and R. Vollgraf, “Texture synthesis with spatial generative adversarial networks,” *arXiv preprint arXiv:1611.08207*, 2016.

[38] R. Szeliski, *Computer vision: Algorithms and applications*. Springer Science & Business Media, 2010.

[39] B. Ghanem and N. Ahuja, “Maximum margin distance learning for dynamic texture recognition,” in *Computer Vision–ECCV 2010*, pp. 223–236, Springer, 2010.

[40] J. Chen, G. Zhao, M. Salo, E. Rahtu, and M. Pietikäinen, “Automatic dynamic texture segmentation using local descriptors and optical flow,” *Image Processing, IEEE Transactions on*, vol. 22, no. 1, pp. 326–339, 2013.

[41] X. Qi, C.-G. Li, G. Zhao, X. Hong, and M. Pietikäinen, “Dynamic texture and scene classification by transferring deep image features,” *Neurocomputing*, vol. 171, pp. 1230–1241, 2016.

[42] B. Julesz, “Visual pattern discrimination,” *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 84–92, 1962.

[43] B. Julesz, “Textons, the elements of texture perception, and their interactions,” *Nature*, vol. 290, no. 5802, pp. 91–97, 1981.

[44] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," *JOSA A*, vol. 2, no. 7, pp. 1160–1169, 1985.

[45] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *International Journal of Computer Vision*, vol. 51, no. 2, pp. 91–109, 2003.

[46] C.-H. Peh and L.-F. Cheong, "Synergizing spatial and temporal texture," *Image Processing, IEEE Transactions on*, vol. 11, no. 10, pp. 1179–1191, 2002.

[47] K. G. Derpanis and R. P. Wildes, "Dynamic texture recognition based on distributions of spacetime oriented structure," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 191–198, IEEE, 2010.

[48] T. Randen and J. H. Husoy, "Filtering for texture classification: A comparative study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 4, pp. 291–310, 1999.

[49] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," *arXiv preprint arXiv:1704.05796*, 2017.

[50] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, pp. 1–42, 2014.

[51] V. Andrearczyk and P. F. Whelan, "Deep learning for biomedical texture image analysis," in *Irish Machine Vision & Image Processing Conference proceedings IMVIP 2016*, 2016.

[52] V. Andrearczyk and P. F. Whelan, "Convolutional neural network on three orthogonal planes for dynamic texture classification," *arXiv preprint arXiv:1703.05530*, 2017.

[53] V. Andrearczyk and P. F. Whelan, "Texture segmentation with fully convolutional networks," *arXiv preprint arXiv:1703.05230*, 2017.

[54] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 4, pp. 532–550, 1987.

[55] H. Voorhees and T. Poggio, "Detecting textons and texture boundaries in natural images," in *Proceedings of the First International Conference on Computer Vision*, vol. 59, pp. 250–258, 1987.

[56] D. Blostein and N. Ahuja, "Shape from texture: Integrating texture-element extraction and surface estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 12, pp. 1233–1251, 1989.

[57] M. Tuceryan and A. K. Jain, "Texture segmentation using voronoi polygons," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 2, pp. 211–216, 1990.

[58] F. Tomita and S. Tsuji, *Computer analysis of visual textures*, vol. 102. Springer Science & Business Media, 1990.

[59] L. S. Davis, S. A. Johns, and J. Aggarwal, "Texture analysis using generalized co-occurrence matrices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 3, pp. 251–259, 1979.

[60] R. M. Haralick, "Statistical and structural approaches to texture," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 786–804, 1979.

[61] T.-H. Hong, C. R. Dyer, and A. Rosenfeld, "Texture primitive extraction using an edge-based approach," tech. rep., DTIC Document, 1979.

[62] R. L. Kashyap, R. Chellappa, and A. Khotanzad, "Texture classification using features derived from random field models," *Pattern Recognition Letters*, vol. 1, no. 1, pp. 43–50, 1982.

[63] F. S. Cohen, Z. Fan, and M. A. Patel, "Classification of rotated and scaled textured images using Gaussian Markov random field models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 2, pp. 192–202, 1991.

[64] N. Cressie, *Statistics for spatial data*. John Wiley & Sons, 2015.

[65] J. Mao and A. K. Jain, "Texture classification and segmentation using multiresolution simultaneous autoregressive models," *Pattern Recognition*, vol. 25, no. 2, pp. 173–188, 1992.

[66] B. B. Mandelbrot and R. Pignoni, "The fractal geometry of nature," 1983.

[67] Y. Fisher, M. McGuire, R. F. Voss, M. F. Barnsley, R. L. Devaney, B. B. Mandelbrot, H.-O. Peitgen, and D. Saupe, *The science of fractal images*. Springer Science & Business Media, 2012.

[68] J. M. Keller, S. Chen, and R. M. Crownover, "Texture description and segmentation through fractal geometry," *Computer Vision, Graphics, and Image Processing*, vol. 45, no. 2, pp. 150–166, 1989.

[69] N. Sarkar and B. Chaudhuri, "An efficient differential box-counting approach to compute fractal dimension of image," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 1, pp. 115–120, 1994.

[70] A. P. Pentland, "Fractal-based description of natural scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 661–674, 1984.

[71] K. C. Clarke, "Computation of the fractal dimension of topographic surfaces using the triangular prism surface area method," *Computers & Geosciences*, vol. 12, no. 5, pp. 713–722, 1986.

[72] M. M. Galloway, "Texture analysis using gray level run lengths," *Computer Graphics and Image Processing*, vol. 4, no. 2, pp. 172–179, 1975.

[73] G. Thibault, B. Fertil, C. Navarro, S. Pereira, P. Cau, N. Levy, J. Sequeira, and J.-L. Mari, "Shape and texture indexes application to cell nuclei classification," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 27, no. 01, p. 1357002, 2013.

[74] H. Kaizer, "A quantification of textures on aerial photographs," *Tech. Note*, vol. 121, 1955.

[75] A. Rosenfeld and A. Kak, *Digital picture processing*. Academic Press, 1982.

[76] K. I. Laws, "Rapid texture identification," in *24th annual technical symposium*, pp. 376–381, International Society for Optics and Photonics, 1980.

[77] W. T. Freeman, E. H. Adelson, *et al.*, "The design and use of steerable filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891–906, 1991.

[78] M. Unser and M. Eden, "Multiresolution feature extraction and selection for texture segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 717–728, 1989.

[79] V. Ojansivu and J. Heikkilä, "Blur insensitive texture classification using local phase quantization," in *International Conference on Image and Signal Processing*, pp. 236–243, Springer, 2008.

[80] A. C. Bovik, M. Clark, and W. S. Geisler, "Multichannel texture analysis using localized spatial filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 1, pp. 55–73, 1990.

[81] J. F. R. Herrera and V. G. Ruiz, "Image coding fundamentals," 2014. Accessed: 2017-03-30.

[82] I. Daubechies, *Ten lectures on wavelets*. SIAM, 1992.

[83] T. Chang and C.-C. Kuo, "Texture analysis and classification with tree-structured wavelet transform," *IEEE Transactions on Image Processing*, vol. 2, no. 4, pp. 429–441, 1993.

[84] N. Kingsbury, "Complex wavelets for shift invariant analysis and filtering of signals," *Applied and Computational Harmonic Analysis*, vol. 10, no. 3, pp. 234–253, 2001.

[85] I. W. Selesnick, R. G. Baraniuk, and N. C. Kingsbury, "The dual-tree complex wavelet transform," *IEEE Signal Processing Magazine*, vol. 22, no. 6, pp. 123–151, 2005.

[86] M. Varma and A. Zisserman, "Texture classification: Are filter banks necessary?," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE computer society conference on*, vol. 2, pp. II–691, IEEE, 2003.

[87] T. Ojala, K. Valkealahti, E. Oja, and M. Pietikäinen, "Texture discrimination with multidimensional distributions of signed gray-level differences," *Pattern Recognition*, vol. 34, no. 3, pp. 727–739, 2001.

[88] M. Pietikäinen, T. Ojala, and Z. Xu, "Rotation-invariant texture classification using feature distributions," *Pattern Recognition*, vol. 33, no. 1, pp. 43–52, 2000.

[89] Z. Guo, L. Zhang, and D. Zhang, "A completed modeling of local binary pattern operator for texture classification," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1657–1663, 2010.

[90] L. Liu, L. Zhao, Y. Long, G. Kuang, and P. Fieguth, "Extended local binary patterns for texture classification," *Image and Vision Computing*, vol. 30, no. 2, pp. 86–99, 2012.

[91] T. Ahonen, J. Matas, C. He, and M. Pietikäinen, "Rotation invariant image description with local binary pattern histogram Fourier features," *Image Analysis*, pp. 61–70, 2009.

[92] X. Tan and B. Triggs, "Enhanced local texture feature sets for face recognition under difficult lighting conditions," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1635–1650, 2010.

[93] L. Liu, P. Fieguth, X. Wang, M. Pietikäinen, and D. Hu, "Evaluation of LBP and Deep Texture Descriptors with a New Robustness Benchmark," in *European Conference on Computer Vision*, pp. 69–86, Springer, 2016.

[94] B. Zhang, Y. Gao, S. Zhao, and J. Liu, "Local derivative pattern versus local binary pattern: face recognition with high-order local pattern descriptor," *IEEE Transactions on Image Processing*, vol. 19, no. 2, pp. 533–544, 2010.

[95] M. Heikkilä, M. Pietikäinen, and C. Schmid, "Description of interest regions with local binary patterns," *Pattern Recognition*, vol. 42, no. 3, pp. 425–436, 2009.

[96] J. Chen, S. Shan, C. He, G. Zhao, M. Pietikäinen, X. Chen, and W. Gao, "WLD: A robust local image descriptor," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1705–1720, 2010.

[97] J. Sivic, A. Zisserman, *et al.*, "Video google: A text retrieval approach to object matching in videos.," in *Computer Vision, 2003. ICCV 2003. Ninth IEEE International Conference on*, vol. 2, pp. 1470–1477, 2003.

[98] M. Varma and A. Zisserman, "A statistical approach to texture classification from single images," *International Journal of Computer Vision*, vol. 62, no. 1-2, pp. 61–81, 2005.

[99] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3606–3613, 2014.

[100] G. Sharma, S. Ul Hussain, and F. Jurie, "Local higher-order statistics (LHS) for texture categorization and facial analysis," in *European Conference on Computer Vision*, pp. 1–12, Springer, 2012.

[101] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, pp. 1–2, Prague, 2004.

[102] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *European Conference on Computer Vision*, pp. 404–417, Springer, 2006.

[103] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2010.

[104] J.-H. Lee, M.-Y. Wu, and H.-T. Kuo, "Evaluation of robust feature descriptors for texture classification," *Evaluation*, vol. 1, p. 11046.

[105] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[106] Y. Ke and R. Sukthankar, “PCA-SIFT: A more distinctive representation for local image descriptors,” in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, pp. II–II, IEEE, 2004.

[107] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF: Binary robust independent elementary features,” in *European Conference on Computer Vision*, pp. 778–792, Springer, 2010.

[108] S. Lazebnik, C. Schmid, and J. Ponce, “A sparse texture representation using local affine regions,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1265–1278, 2005.

[109] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass support vector machines,” *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[110] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, “Image classification with the fisher vector: Theory and practice,” *International journal of Computer Vision*, vol. 105, no. 3, pp. 222–245, 2013.

[111] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[112] B. Caputo, E. Hayman, and P. Mallikarjuna, “Class-specific material categorisation,” in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2, pp. 1597–1604, IEEE, 2005.

[113] K. J. Dana, B. Van Ginneken, S. K. Nayar, and J. J. Koenderink, “Reflectance and texture of real-world surfaces,” *ACM Transactions On Graphics (TOG)*, vol. 18, no. 1, pp. 1–34, 1999.

[114] N. Paragios and R. Deriche, “Geodesic active regions and level set methods for supervised texture segmentation,” *International Journal of Computer Vision*, vol. 46, no. 3, pp. 223–247, 2002.

[115] W. Byeon and T. M. Breuel, “Supervised texture segmentation using 2D LSTM networks,” in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 4373–4377, IEEE, 2014.

[116] N. Mevenkamp and B. Berkels, “Variational multi-phase segmentation using high-dimensional local features,” in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1–9, IEEE, 2016.

[117] X. Liu and D. Wang, “Image and texture segmentation using local spectral histograms,” *IEEE Transactions on Image Processing*, vol. 15, no. 10, pp. 3066–3077, 2006.

[118] J. Yuan, D. Wang, and A. M. Cheriyadat, “Factorization-based texture segmentation,” *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3488–3497, 2015.

[119] C. Bouman and B. Liu, “Multiple resolution segmentation of textured images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 2, pp. 99–113, 1991.

[120] Z. Kato, T.-C. Pong, and J. C.-M. Lee, “Color image segmentation and parameter estimation in a markovian framework,” *Pattern Recognition Letters*, vol. 22, no. 3, pp. 309–321, 2001.

[121] P. C. Chen and T. Pavlidis, “Segmentation by texture using a co-occurrence matrix and a split-and-merge algorithm,” *Computer Graphics and Image Processing*, vol. 10, no. 2, pp. 172–182, 1979.

[122] S. Basu, M. Karki, S. Mukhopadhyay, S. Ganguly, R. Nemani, R. DiBiano, and S. Gayaka, “A theoretical analysis of deep neural networks for texture classification,” pp. 992–999, 2016.

[123] Y. Song, Q. Li, D. Feng, J. J. Zou, and W. Cai, “Texture image classification with discriminative neural networks,” *Computational Visual Media*, vol. 2, no. 4, pp. 367–377, 2016.

[124] F. H. C. Tivive and A. Bouzerdoum, “Texture classification using convolutional neural networks,” in *TENCON 2006-2006 IEEE Region 10 Conference*, pp. 1–4, IEEE, 2006.

[125] L. G. Hafemann, L. S. Oliveira, and P. Cavalin, “Forest species recognition using deep convolutional neural networks,” in *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pp. 1103–1107, IEEE, 2014.

[126] L. G. Hafemann, L. S. Oliveira, P. R. Cavalin, and R. Sabourin, “Transfer learning between texture classification tasks using Convolutional Neural Networks,” in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, 2015.

[127] W. Byeon, M. Liwicki, and T. M. Breuel, “Texture classification using 2D LSTM networks,” in *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pp. 1144–1149, IEEE, 2014.

[128] J. Bruna and S. Mallat, “Invariant scattering convolution networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[129] L. Sifre and S. Mallat, “Rotation, scaling and deformation invariant scattering for texture discrimination,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1233–1240, 2013.

[130] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, “PCANet: A simple deep learning baseline for image classification?,” *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

[131] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition.,” in *Icml*, vol. 32, pp. 647–655, 2014.

[132] T.-Y. Lin, A. RoyChowdhury, and S. Maji, “Bilinear CNN models for fine-grained visual recognition,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1449–1457, 2015.

[133] H. Zhang, J. Xue, and K. Dana, “Deep TEN: Texture Encoding Network,” *arXiv preprint arXiv:1612.02844*, 2016.

[134] D. Marcos, M. Volpi, and D. Tuia, "Learning rotation invariant convolutional filters for texture classification," *arXiv preprint arXiv:1604.06720*, 2016.

[135] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 262–270, 2015.

[136] B. Afsari, R. Chaudhry, A. Ravichandran, and R. Vidal, "Group action induced distances for averaging and clustering linear dynamical systems with applications to the analysis of dynamic scenes," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2208–2215, IEEE, 2012.

[137] D. Tiwari and V. Tyagi, "Dynamic texture recognition using multiresolution edge-weighted local structure pattern," *Computers & Electrical Engineering*, 2016.

[138] E. Rahtu, J. Heikkilä, V. Ojansivu, and T. Ahonen, "Local phase quantization for blur-insensitive image analysis," *Image and Vision Computing*, vol. 30, no. 8, pp. 501–512, 2012.

[139] F. Yang, G.-S. Xia, G. Liu, L. Zhang, and X. Huang, "Dynamic texture recognition by aggregating spatial and temporal features via ensemble SVMs," *Neurocomputing*, vol. 173, pp. 1310–1321, 2016.

[140] R. Péteri and D. Chetverikov, "Dynamic texture recognition using normal flow and texture regularity," in *Pattern Recognition and Image Analysis*, pp. 223–230, Springer, 2005.

[141] S. Fazekas and D. Chetverikov, "Dynamic texture recognition using optical flow features and temporal periodicity," in *Content-Based Multimedia Indexing, 2007. CBMI'07. International Workshop on*, pp. 25–32, IEEE, 2007.

[142] A. Ravichandran, R. Chaudhry, and R. Vidal, "Categorizing dynamic textures using a bag of dynamical systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 2, pp. 342–353, 2013.

[143] K. Fujita and S. K. Nayar, "Recognition of dynamic textures using impulse responses of state variables," in *Proc. Third International Workshop on Texture Analysis and Synthesis (Texture 2003)*, pp. 31–36, 2003.

[144] Q. Yu-long and W. Fu-shan, "Dynamic texture classification based on dual-tree complex wavelet transform," in *Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on*, pp. 823–826, IEEE, 2011.

[145] W. N. Gonçalves, B. B. Machado, and O. M. Bruno, "Spatiotemporal Gabor filters: A new method for dynamic texture recognition," *arXiv preprint arXiv:1201.3612*, 2012.

[146] H. Ji, X. Yang, H. Ling, and Y. Xu, "Wavelet domain multifractal analysis for static and dynamic texture classification," *Image Processing, IEEE Transactions on*, vol. 22, no. 1, pp. 286–299, 2013.

[147] C. Feichtenhofer, A. Pinz, and R. Wildes, "Bags of spacetime energies for dynamic scene recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2681–2688, 2014.

[148] G. Zhao and M. Pietikäinen, "Dynamic texture recognition using volume local binary patterns," in *Dynamical Vision*, pp. 165–177, Springer, 2007.

[149] D. Tiwari and V. Tyagi, "Improved Weber's law based local binary pattern for dynamic texture recognition," *Multimedia Tools and Applications*, pp. 1–18, 2016.

[150] G. Zhao, T. Ahonen, J. Matas, and M. Pietikainen, "Rotation-invariant image and video description with local binary pattern features," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 1465–1477, 2012.

[151] V. Andrearczyk and P. F. Whelan, "Dynamic texture classification using combined co-occurrence matrices of optical flow," in *Irish Machine Vision & Image Processing Conference proceedings IMVIP 2015*, 2015.

[152] A. Ramirez Rivera and O. Chae, "Spatiotemporal directional number transitional graph for dynamic texture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2146–2152, 2015.

[153] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[154] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," *arXiv preprint arXiv:1412.0767*, 2014.

[155] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[156] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems*, pp. 568–576, 2014.

[157] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2625–2634, 2015.

[158] J. Shao, C.-C. Loy, K. Kang, and X. Wang, "Slicing convolutional neural network for crowd video understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5620–5628, 2016.

[159] J. Shao, C. C. Loy, K. Kang, and X. Wang, "Crowded scene understanding by deeply learned volumetric slices," *IEEE Transactions on Circuits and Systems for Video Technology*, 2016.

[160] D. Culibrk and N. Sebe, "Temporal dropout of changes approach to convolutional learning of spatio-temporal features," in *Proceedings of the 22nd ACM International Conference on Multimedia*, pp. 1201–1204, ACM, 2014.

[161] X. Yan, H. Chang, S. Shan, and X. Chen, "Modeling video dynamics with deep dynencoder," in *European Conference on Computer Vision*, pp. 215–230, Springer, 2014.

[162] S. R. Arashloo, M. C. Amirani, and A. Noroozi, “Dynamic texture representation using a deep multi-scale convolutional network,” *Journal of Visual Communication and Image Representation*, 2016.

[163] S. Hong, J. Ryu, W. Im, and H. S. Yang, “Recognizing dynamic scenes with deep dual descriptor based on key frames and key segments,” *arXiv preprint arXiv:1702.04479*, 2017.

[164] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[165] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature mbedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[166] G. Kylberg and I.-M. Sintorn, “Evaluation of noise robustness for local binary pattern descriptors in texture classification,” *EURASIP Journal on Image and Video Processing*, vol. 2013, no. 1, pp. 1–20, 2013.

[167] P. L. Paula Filho, L. S. Oliveira, S. Nisgoski, and A. S. Britto Jr, “Forest species recognition using macroscopic images,” *Machine Vision and Applications*, vol. 25, no. 4, pp. 1019–1031, 2014.

[168] J. Martins, L. Oliveira, S. Nisgoski, and R. Sabourin, “A database for automatic classification of forest species,” *Machine Vision and Applications*, vol. 24, no. 3, pp. 567–578, 2013.

[169] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, “Visualizing higher-layer features of a deep network,” *University of Montreal*, vol. 1341, 2009.

[170] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization,” *arXiv preprint arXiv:1506.06579*, 2015.

[171] N. Orlov, L. Shamir, T. Macura, J. Johnston, D. M. Eckley, and I. G. Goldberg, “WND-CHARM: Multi-purpose image classification using compound image transforms,” *Pattern Recognition Letters*, vol. 29, no. 11, pp. 1684–1693, 2008.

[172] V. Uhlmann, S. Singh, and A. E. Carpenter, “CP-CHARM: Segmentation-free image classification made accessible,” *BMC Bioinformatics*, vol. 17, no. 1, p. 1, 2016.

[173] N. Hervé, A. Servais, E. Thervet, J.-C. Olivo-Marin, and V. Meas-Yedid, “Statistical color texture descriptors for histological images analysis,” in *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 724–727, IEEE, 2011.

[174] H.-L. Huang, M.-H. Hsu, H.-C. Lee, P. Charoenkwan, S.-J. Ho, and S.-Y. Ho, “Prediction of mouse senescence from HE-Stain liver images using an ensemble SVM classifier,” in *Asian Conference on Intelligent Information and Database Systems*, pp. 325–334, Springer, 2013.

[175] L. Shamir, N. Orlov, D. M. Eckley, T. J. Macura, and I. G. Goldberg, "IICBU 2008: A proposed benchmark suite for biological image analysis," *Medical & Biological Engineering & Computing*, vol. 46, no. 9, pp. 943–947, 2008.

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

[177] Y. Xu, Y. Quan, H. Ling, and H. Ji, "Dynamic texture classification using dynamic fractal analysis," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 1219–1226, IEEE, 2011.

[178] E. Norouznezhad, M. T. Harandi, A. Bigdeli, M. Baktash, A. Postula, and B. C. Lovell, "Directional space-time oriented gradients for 3d visual pattern analysis," in *Computer Vision–ECCV 2012*, pp. 736–749, Springer, 2012.

[179] J. Ren, X. Jiang, and J. Yuan, "Dynamic texture recognition using enhanced LBP features.," in *ICASSP*, pp. 2400–2404, 2013.

[180] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2011 VOC2011 results." http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html, 2016.

[181] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *AISTATS*, vol. 9, pp. 249–256, 2010.

[182] M. Haindl and S. Mikes, "The prague texture segmentation datagenerator and benchmark." http://mosaic.utia.cas.cz/index.php?act=view_res, 2008. [Online; accessed 25-January-2017].

[183] C. Panagiotakis, I. Grinias, and G. Tziritas, "Natural image segmentation based on tree equipartition, bayesian flooding and region merging," *IEEE Transactions on Image Processing*, vol. 20, no. 8, pp. 2276–2287, 2011.

[184] I. G. Y. Bengio and A. Courville, "Deep learning." Book in preparation for MIT Press, 2016.

[185] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[186] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 427–436, 2015.

[187] H. Zheng, L. Fang, M. Ji, M. Strese, Y. Özer, and E. Steinbach, "Deep learning for surface material classification using haptic and visual information," *IEEE Transactions on Multimedia*, vol. 18, no. 12, pp. 2407–2416, 2016.

[188] T. Rao, M. Xu, and D. Xu, "Learning multi-level deep representations for image emotion classification," *arXiv preprint arXiv:1611.07145*, 2016.

[189] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[190] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[191] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[192] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[193] D. Kingma and J. Ba, "ADAM: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[194] G. E. Hinton, "Learning translation invariant recognition in a massively parallel networks," in *International Conference on Parallel Architectures and Languages Europe*, pp. 1–13, Springer, 1987.

[195] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[196] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[197] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[198] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[199] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[200] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[201] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[202] M. A. Ranzato, C. Poultney, S. Chopra, Y. L. Cun, *et al.*, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems*, pp. 1137–1144, 2006.

[203] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[204] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, no. Aug, pp. 115–143, 2002.

[205] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2528–2535, IEEE, 2010.

[206] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[207] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 4898–4906, 2016.

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

[209] M. A. Ranzato, F. J. Huang, Y.-L. Boureau, Y. LeCun, *et al.*, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *2007 IEEE conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2007.

[210] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[211] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*, pp. 818–833, Springer, 2014.

[212] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

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

[214] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 447–456, 2015.

[215] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[216] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[217] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Advances in Neural Information Processing Systems*, pp. 550–558, 2016.

[218] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[219] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[220] F. Chollet, “Deep learning with separable convolutions,” *arXiv preprint arXiv:1610.02357*, 2016.

[221] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.

[222] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang, “Recent advances in convolutional neural networks,” *arXiv preprint arXiv:1512.07108*, 2015.

Appendix A

Introduction to deep learning and convolutional neural networks

A.1 Introduction

A.1.1 Overview

This appendix introduces neural networks and deep learning methods followed by Convolutional Neural Networks (CNNs). It seeks to explain the main concepts used in the thesis as well as some of the most popular architectures and advances. An effort was made to keep the content concise while giving an overview and defining important notions used in the thesis. Only a small portion of the constantly expanding deep learning and CNN research is therefore covered. Many references are provided throughout the chapter for the readers who seek more information on specific topics. In particular, more details on neural networks can be found in [111] while deep learning methods and CNNs are covered in detail in [184].

A.1.2 Definitions

Neural network

Neural networks, also known as Artificial Neural Networks (ANNs), are biologically-inspired machine learning approaches which can learn how to perform tasks from observed data. In an ANN, sets of neurons are organised in interconnected layers. The connections between neurons are called weights (the equivalent of synapses in a biological neural network) and can be trained automatically to infer a function with an optimisation method. A key difference from classic machine learning approaches is the possibility of replacing the hand-crafted features by powerful learning algorithms.

Deep learning

Deep learning is a machine learning approach to train deep neural networks, i.e. ANNs with many layers. Deep neural networks learn multiple levels of data representation in order to build a hierarchy of concepts. It is based on a cascade of layers which compute linear and non-linear transformations of the input data to build an increasing level of abstraction of the data. Abstraction refers in this document to the invariance of features to variations of the same concept [189]. Similarly to machine learning algorithms, it can be supervised, semi-supervised or unsupervised.

A.1.3 Motivation

Simple artificial intelligence algorithms enable computers to compute tasks that are difficult for humans (e.g. playing chess, weather or stock market forecast). These tasks are formal and restricted in the level of abstraction of representation required to perform them. However, other tasks that are very easily computed by humans and sometimes animals (e.g. vision, speech, speech understanding and reading) are extremely difficult to implement with classic machine learning algorithms. These difficulties arise from the size of input data and the fact that the tasks are more informal and require a higher level of abstraction. Traditional machine learning algorithms use pre-defined (or hand-crafted) features to infer a function to perform a formal task from a small training dataset. The idea of neural networks is to replace these hand-crafted features by trainable parameters which are able to learn complex and abstract concepts. In theory, a shallow network (generally less than three layers) can approximate arbitrarily well any continuous function on a compact domain. However, the width of the network (number of neurons per layer) must be large and the optimisation of such network is difficult. Deep networks can infer complex functions more efficiently with fewer parameters and are easier to train. The cascades of layers in deep neural networks build representations of increasing complexity. These representations are expressed as an abstraction of previous simpler representations in the network. With the development of the computational power of computers (mainly GPUs), the breakthrough of deep learning algorithms has been to train deep neural networks with very large databases. Deeper architectures, larger databases, and new learning methods keep improving the accuracy of deep learning on the currently most challenging artificial intelligence tasks. Some of the most important deep learning advances are found in computer vision and natural language processing.

A.2 Neural networks

This section introduces the artificial neurons and shallow neural networks. It provides an explanation of the backpropagation algorithm used to train neural networks as well as several important network architectures.

A.2.1 The neuron

An artificial neuron (called neuron for simplicity) is a mathematical function inspired by biological neurons. Neural networks are designed by connecting multiple neurons together.

Linear neuron

The first artificial neuron (also known as a unit) was developed in 1957 [190] in a single layer network called perceptron. This neuron performs a very simple supervised binary classifier. An example of linear neuron used in a perceptron is shown in Figure A.1. The neuron represented by a circle takes a vector input $\mathbf{x}$ and computes a single binary value output as:

$$h(\mathbf{x}) = \begin{cases} 1 \text{ if } \sum_{i=0}^{m} w_i x_i + b > 0 \\ 0 \text{ otherwise} \end{cases} \tag{A.1}$$

where $\mathbf{w}$ is a vector of m real-valued weights, m is also the number of inputs and b is the bias. The weights and biases are interchangeably called (trainable) parameters. The i^{th} input value x_i is "connected" to the neuron by the multiplication to the weight w_i. The bias shifts the decision boundary of a neuron away from the origin without being connected to any input value, similar to a bias b that shifts a line $l = ax + b$ regardless the value of x (see Figure A.2).

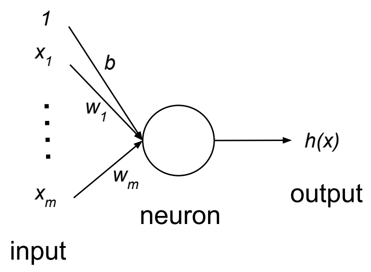


Figure A.1: A perceptron linear neuron: first developed artificial neuron.

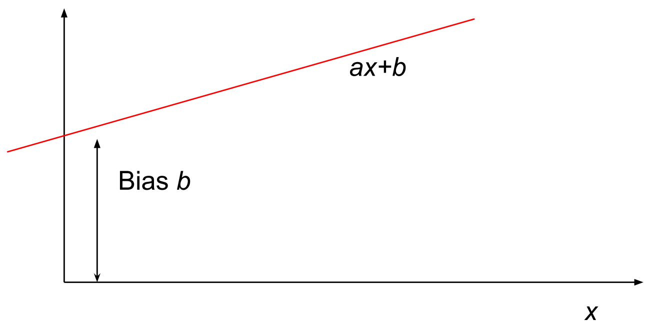


Figure A.2: An illustration of a weight a and bias b in a line equation example $ax+b$.

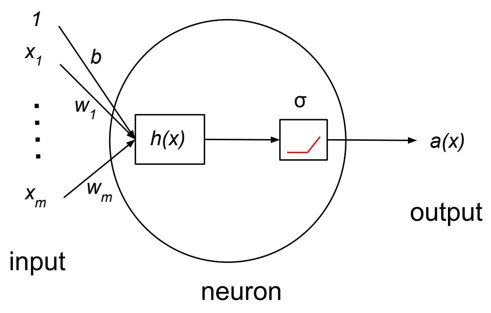


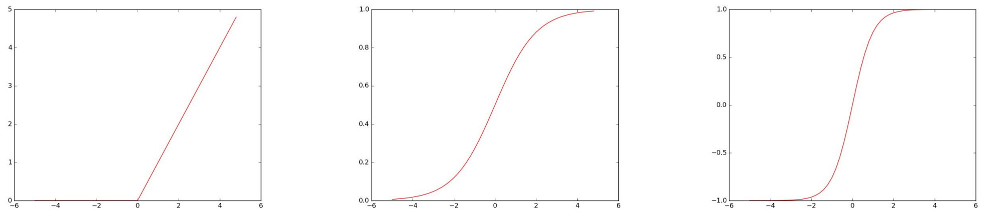
Figure A.3: A recent neuron with activation function.

More recent neurons with non-linearity

More recent neurons can be represented by Figure A.3. They compute a real value as opposed to the binary output of the linear neuron. These neurons introduce a non-linearity, by the “activation” of the linear weighted combination of inputs, essential in neural networks with multiple layers as discussed in Section A.2.2. A commonly used neuron computes a non-linear weighted sum as:

$$a(\mathbf{x}) = \sigma(h(\mathbf{x})) = \sigma(\sum_{i=1}^{m} w_i x_i + b) \tag{A.2}$$

where $\mathbf{w}$ are the weights and $\mathbf{x}$ the inputs. Both are vectors of size m. Parameter b is the bias and σ is a non-linear function inspired by the “firing” of biological neurons. The use of non-linearity is necessary when using multiple layers of neurons which will be explained later. In Figure A.3, the neuron encapsulates the computation of $h(\mathbf{x})$ and its activation with a non-linear function $a(\mathbf{x}) = \sigma(h(\mathbf{x}))$. In the rest of this section, neurons are illustrated with a single circle which implies that the hidden neurons compute the activation $a(\mathbf{x})$. Several non-linear functions have been used in neural networks. The most frequently used and popular ones are the following.



(a) ReLU **(b)** Sigmoid **(c)** tanh

Figure A.4: A commonly used non-linear activation functions.

- Rectified Linear Unit (ReLU) [16]:

$$\sigma(x) = max(0,x) \tag{A.3}$$

- Sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{A.4}$$

- Hyperbolic tangent:

$$\sigma(x) = tanh(x) \tag{A.5}$$

These activation functions are applied element-wise to pre-activation vectors when denoted $\sigma(\mathbf{x})$. Note that ReLU is generally used in deep networks for avoiding the vanishing gradients problem as explained in Section A.3.2.

A.2.2 Artificial neural network architecture

Neurons are organised in an ANN into multiple layers. A basic example of such network is the Multi Layer Perceptron (MLP) [1]. An MLP consists of an input layer (bottom of the network), an output layer (top of the network) and at least one hidden layer between them. The depth of a network generally refers to the number of layers in the network excluding the input layer, i.e. number of hidden layers plus one output layer. An example of a three layer MLP is illustrated in Figure A.5. A MLP is fully connected, i.e. each neuron of a given hidden layer is connected to all the neurons of the previous and next layers.

Non-linearity

In the given example, similar to most current neural network architectures, each hidden neuron computes its output based on Eq. A.2. Note that in Figure A.5, the biases are omitted for simplicity although each neuron is connected to a bias of value 1 with a weight *b* as shown in Figure A.1. If only linear neurons were used, a network

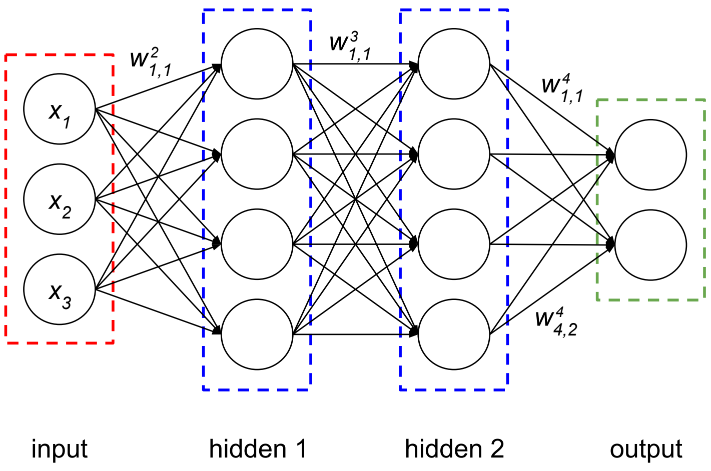


Figure A.5: A three layer Multi Layer Perceptron.

of any depth could be reduced to a single layer perceptron network which would perform a linear regression. Instead, the output of the neurons are "activated" with a non-linear function σ and more complex non-linear functions can be achieved.

Notations

To generalise to other possibly deeper architectures than the one in Figure A.5, an $L-1$ layer network (L layers including the input) is considered as illustrated in Figure A.6.

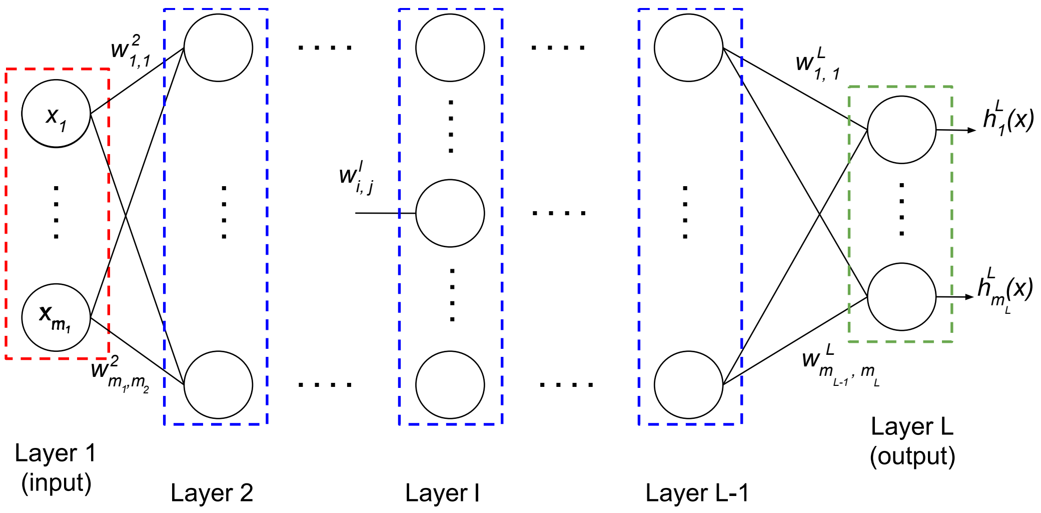


Figure A.6: The architecture of an $L-1$ layers neural network (L layers including the input).

.

In this architecture, h_j^l is the output of neuron j at layer l (with l=1 being the input layer) before activation with:

$$h_j^l = \sum_{i=1}^{m_{l-1}} w_{ij}^l a_i^{l-1} + b_j^l \tag{A.6}$$

where a_i^{l-1} is the output of the activation function $\sigma(h_i^{l-1})$, w_{ij}^l is the weight connecting a_i^{l-1} to the j^{th} neuron at layer l and m_l is the number of neurons at layer l. With these notations the output of a hidden neuron is given by:

$$a_j^l = \sigma(h_j^l) = \sigma(\sum_{i=1}^{m_{l-1}} w_{ij}^l a_i^{l-1} + b_j^l) \tag{A.7}$$

A.2.3 Training a neural network: Backpropagation

Two tasks are commonly studied with neural networks, namely classification and regression. Both are discriminative models which learn to estimate a class label (classification) or a prediction value (regression) knowing an input sample, as opposed to generative models mentioned in Section A.2.6. The common scenario of classification with an ANN will be described in the following sections. In this scenario, the output of the last layer is a vector of size N, where $N = m_L$ is the number of classes. The values of the output neurons represent a class score or a (non-normalised) probability of the input layer to belong to a certain class. Labelled training data is fed into the network to infer a function whose outputs are close to the given labels and ideally generalises to unknown (unlabelled) data. This approach is called “supervised learning”. In the example given in Figure A.5, the network can be used for a binary classification as the output layer contains two neurons.

As mentioned previously in the appendix, a key factor of success of neural networks is the automatic learning of the weights (and biases). Backpropagation is a training method to iteratively update the weights in order to minimise the error between known training labels and the predictions of the network. To this end, the error is propagated backwards through the network by calculating the gradients of a defined loss function w.r.t. all the parameters. These gradients are a measure of how the error will change when each weight is varied individually. An optimisation function then uses these gradients to update the weights and reduce the error. The backpropagation algorithm can be summarised as a succession of the following operations: forward pass, error calculation, backpropagation of the gradients and optimisation of the weights. These operations will be discussed in more details as well as other concepts to efficiently train neural networks. Note that the forward and backward passes are computed many times with various input samples during training.

Forward pass

In the forward pass, the outputs of all the neurons are computed from the input layer to the output layer. The hidden neurons compute $a_j^l = \sigma(h(\mathbf{a}^{l-1}))$ and the output of the network is $f(\mathbf{x}) = \mathbf{h}^L = h(\mathbf{a}^{L-1})$.

Error

The output of the last layer as mentioned previously is a vector of real values that represent class scores. With a training set or subset of input vectors $\mathbf{x}^{(k)} \in \mathbf{X}$, k in the range $(1,K)$ and associated ground truth vectors (also known as target or desired output) $\mathbf{y}^{(k)} = f(\mathbf{x}^{(k)}) \in \mathbf{Y}$, the network is trained so that it computes a function $f(\mathbf{X})$ which maps $\mathbf{X}$ to $\mathbf{Y}$. This function should also generalise well to unknown input data. Overfitting and underfitting will be discussed later in this appendix. To know how well the network is performing, the errors between the ground truths $\mathbf{y}^{(k)}$ of the given inputs $\mathbf{x}^{(k)}$ and the predictions of the network $f(\mathbf{x}^{(k)})$ are calculated and averaged across the multiple K samples. Common error functions (also known as loss, cost or objective functions) include the *mean squared* and the *cross entropy* errors.

The mean squared error (also known as Euclidean loss) is often used for real-valued regression tasks but can also be used for classification. The error for a given input $\mathbf{x}^{(k)}$ is computed as:

$$E^{(k)} = \frac{1}{2}\sum_{n=1}^{N}(h_n^{L(k)} - y_n^{(k)})^2 \tag{A.8}$$

The error across the set of inputs $\mathbf{X}$ is given by:

$$E = \frac{1}{2K}\sum_{k=1}^{K}\sum_{n=1}^{N}(h_n^{L(k)} - y_n^{(k)})^2 \tag{A.9}$$

The multi-class cross entropy (also known as logistic loss) is often used for predicting targets interpreted as probabilities. The cross entropy calculates the difference between a predicted probability vector $\hat{p}(\mathbf{x})$ and its ground truth $p(\mathbf{x})$. A vector of predictions of the network $f(\mathbf{x}) = \mathbf{h}^L$ is generally interpreted (or squashed) as a probability vector using the following softmax function.

$$\hat{p}_n^{(k)} = \frac{e^{h_n^{L(k)}}}{\sum_{j=1}^{N} e^{h_j^{L(k)}}} \text{ for } n = 1, ..., N \tag{A.10}$$

where $h_j^{L(k)}$ is the j^{th} value of the output vector of the network with input $\mathbf{x}^{(k)}$. The values of the probability prediction vector $\hat{\mathbf{p}}^{(k)}$ are in the range [0,1] and sum to 1.

The cross entropy loss is then computed as follows:

$$E^{(k)} = -\sum_{n=1}^{N} p_n^{(k)} \log \hat{p}_n^{(k)} \tag{A.11}$$

The values of p_n being constant, minimising Eq. A.11 is equivalent to minimising the function:

$$-\sum_{n=1}^{N} p_n^{(k)} \log \hat{p}_n^{(k)} + \sum_{n=1}^{N} p_n^{(k)} \log p_n^{(k)} \tag{A.12}$$

Doing so a new error function is obtained with convenient partial derivatives as will be shown in the next section.

$$E^{(k)} = -\sum_{n=1}^{N} p_n^{(k)} \log \left(\frac{\hat{p}_n^{(k)}}{p_n^{(k)}} \right) \tag{A.13}$$

The error across the set of inputs **X** is:

$$E = -\frac{1}{K} \sum_{k=1}^{K} \sum_{n=1}^{N} p_n^{(k)} \log \left(\frac{\hat{p}_n^{(k)}}{p_n^{(k)}} \right) \tag{A.14}$$

Backpropagation of the error

With the error of the network's output calculated, the gradients (partial derivatives) of the error w.r.t. all the parameters can now be derived. The gradients are calculated in this section with the softmax cross entropy loss (Eq. A.10 and A.13). A similar calculation can easily be derived with other differentiable loss functions. Using the chain rule, the partial derivative of E w.r.t. the parameters of the last layer can be expressed as follows:

$$\frac{\partial E^{(k)}}{\partial w_{ij}^{L}} = \frac{\partial E^{(k)}}{\partial h_j^{L(k)}} \frac{\partial h_j^{L(k)}}{\partial w_{ij}^{L}} \tag{A.15}$$

For simplification of the equations, δ^l (not to be confused with the Kronecker delta) is introduced as follows:

$$\delta_j^{l(k)} = \frac{\partial E^{(k)}}{\partial h_j^{l(k)}} \tag{A.16}$$

δ^l is sometimes referred to as the *errors* at a certain layer l. The gradients equation becomes:

$$\frac{\partial E^{(k)}}{\partial w_{ij}^{L}} = \delta_j^{L(k)} \frac{\partial h_j^{L(k)}}{\partial w_{ij}^{L}} \tag{A.17}$$

The chain rule can be applied for the calculation of $\delta_j^{L(k)}$ by summing over all the output neurons.

$$\delta_j^{L(k)} = \frac{\partial E^{(k)}}{\partial h_j^{L(k)}} = \sum_{n=1}^{N} \frac{\partial E^{(k)}}{\partial \hat{p}_n^{(k)}} \frac{\partial \hat{p}_n^{(k)}}{\partial h_j^{L(k)}} \tag{A.18}$$

From the cross-entropy Eq. A.13, the first partial derivative can be expressed as follows:

$$\frac{\partial E^{(k)}}{\partial \hat{p}_n^{(k)}} = -\frac{p_n^{(k)}}{\hat{p}_n^{(k)}} \tag{A.19}$$

And from Eq. A.10, the second term is derived as:

$$\frac{\partial \hat{p}_n^{(k)}}{\partial h_j^{L(k)}} = \begin{cases} \hat{p}_n^{(k)}(1-\hat{p}_j^{(k)}) \text{ if } n = j \\ -\hat{p}_n^{(k)}\hat{p}_j^{(k)} \text{ if } n \neq j \end{cases} \tag{A.20}$$

The above equation can be simplified using the Kronecker delta δ_{nj} (not to be mistaken with the errors δ^l) so that:

$$\frac{\partial \hat{p}_n^{(k)}}{\partial h_j^{L(k)}} = \hat{p}_n^{(k)}(\delta_{nj} - \hat{p}_j^{(k)}) \text{ with } \delta_{nj} = \begin{cases} 1 \text{ if } n = j \\ 0 \text{ if } n \neq j \end{cases} \tag{A.21}$$

Eq. A.18 can be re-written to substitute Eq. A.19 and A.21 as follows:

$$\delta_j^{L(k)} = \frac{\partial E^{(k)}}{\partial \hat{p}_j^{(k)}} \frac{\partial \hat{p}_j^{(k)}}{\partial h_j^{L(k)}} + \sum_{n \neq j} \frac{\partial E^{(k)}}{\partial \hat{p}_n^{(k)}} \frac{\partial \hat{p}_n^{(k)}}{\partial h_j^{L(k)}} \tag{A.22}$$

Which can be simplified as:

$$\delta_j^{L(k)} = \hat{p}_j^{(k)} - p_j^{(k)} \tag{A.23}$$

The partial derivatives of the outputs $h_j^{L(k)}$ w.r.t. the weights can now be easily calculated:

$$\frac{\partial h_j^{L(k)}}{\partial w_{ij}^L} = a_i^{L-1(k)} \tag{A.24}$$

The partial derivative of the error w.r.t. the weights (Eq. A.17) now becomes:

$$\frac{\partial E^{(k)}}{\partial w_{ij}^L} = \delta_j^{L(k)} a_i^{L-1(k)} = (\hat{p}_j^{(k)} - p_j^{(k)}) a_i^{L-1(k)} \tag{A.25}$$

The gradients for the bias b_j^L are also easily computed as:

$$\frac{\partial h_j^{L(k)}}{\partial b_j^L} = 1 \tag{A.26}$$

Having calculated the gradients of the weights at the last layer L, the backpropagation can continue by calculating the gradients of all the weights in the network. For all other layers, the neurons compute their output with Eq. A.7. The weights gradients at layer $l < L$ are calculated as:

$$\frac{\partial E^{(k)}}{\partial w_{ij}^l} = \delta_j^{l(k)} a_j^{l-1(k)} \tag{A.27}$$

where the δ's are:

$$\delta_j^{l(k)} = \sigma'(h_j^{l(k)}) \sum_{n=1}^{m_{l+1}} w_{jn}^{l+1} \delta_n^{l+1(k)} \tag{A.28}$$

where m_{l+1} is the number of output neurons at layer $l+1$ and σ' is the derivative of the activation function, e.g.:

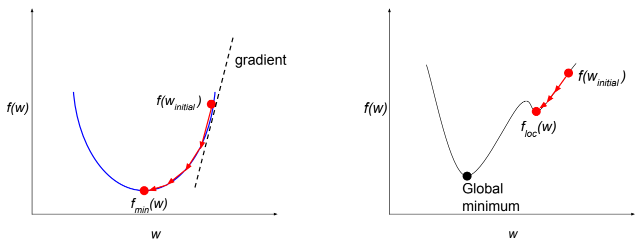
$$\sigma'_{ReLU}(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{A.29}$$

$$\sigma'_{sigmoid}(x) = x(1-x) \tag{A.30}$$

$$\sigma'_{tanh}(x) = 1 - tanh^2(x) \tag{A.31}$$

Stochastic gradient descent

Gradient Descent (GD) (or steepest descent) is an optimisation method to find a local (preferably global) minima of a function. In backpropagation, it is used to iteratively update the weights in order to minimise the error function. Figure A.7a illustrates a gradient descent approach which updates a parameter (or weight) w to minimise a function $f(w)$. Figure A.7b shows the common local minimum problem with GD methods. In neural networks, the optimisation is applied to a much higher dimension with many local minima. In a GD optimisation, all the training samples are used for each update of the weights whereas in Stochastic Gradient Descent (SGD), only one or a small batch of training samples are used for each step. With the often large training sets, the computation time of the GD optimisation can become extremely long as one must compute the outputs, errors, and gradients of all the samples at each iteration. SGD is therefore almost always preferred to GD in neural networks.



(a) global minimum **(b)** local minimum

Figure A.7: A 1D gradient descent minimisation of $f(w)$ with (a) global minimum, (b) local minimum.

The weights are updated using:

$$w_{ij}^{\tau+1} = w_{ij}^{\tau} + \Delta w_{ij}^{\tau} \tag{A.32}$$

where τ is the gradient descent iteration and Δw_{ij} is the weight update. Two variants of SGD are usually used, namely *on-line learning* and *batch learning*.

In on-line learning, the weights are updated for every single input $\mathbf{x}$, i.e. in Eq. A.32 the update is computed as:

$$\Delta w_{ij} = -\eta \frac{\partial E^{(k)}}{\partial w_{ij}} = -\eta \delta_j^l a_j^{l-1(k)} \tag{A.33}$$

where the learning rate η $(0 < \eta < 1)$ is a hyperparameter of the network that determines the amount of change of the weights at every iteration. The learning rate can be either fixed or gradually lower throughout the entire training.

In batch learning, on the other hand, the error is calculated for a batch of K inputs $\mathbf{X}$ and the weights gradients are averaged as:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\frac{\eta}{K} \sum_{k=1}^{K} \delta_j^{l(k)} a_j^{l-1(k)} \tag{A.34}$$

With on-line learning, the stochastic error surface is noisy which helps to escape local minima. Batch learning, by averaging the gradients, removes noise and is more prone to being caught in local minima. Batch training, however, is preferred as it can be implemented very efficiently with modern computers. Moreover, other methods have been developed to escape local minima and these will be described in detail in the following sections.

Momentum

As mentioned before, a common issue with the gradient descent is being trapped in a local minimum far from the desired global minimum as illustrated in Figure A.7b. The momentum [1] is a method which helps to escape from a local minimum by diminishing the fluctuations in weight updates over consecutive iterations. One can think of it as a ball rolling down the slope in the weights space and picking up speed, thus rolling up the opposite slope to some extent when reaching a minima. It also helps to converge faster when the gradients are very small. The weight update is based on the accumulation of first-order information (gradients) over time and Eq. A.34 becomes:

$$\Delta w_{ij}^{\tau} = -\eta \frac{\partial E^{\tau}}{\partial w_{ij}^{\tau}} + m \Delta w_{ij}^{\tau-1} \tag{A.35}$$

where the momentum $m \in [0,1]$ is another hyperparameter which determines the contribution of earlier gradients to the weight change.

Adagrad

The adaptive gradient (Adagrad) [191] is a method to adapt the learning rate in SGD to the trainable parameters, avoiding the manual tuning of the learning rate used for all the weights in the classic SGD. The learning rate used in Eq. A.32 and A.34 is calculated at each time step and for each weight based on the previous squared gradients (second-order information) and becomes:

$$w_{ij}^{\tau+1} = w_{ij}^{\tau} - \frac{\eta}{\sqrt{v_{ij}^{\tau} + \varepsilon}} \frac{\partial E^{\tau}}{\partial w_{ij}^{\tau}} \tag{A.36}$$

where v_{ij}^{τ} is the sum of the squared gradients $(\frac{\partial E^{\tau}}{\partial w_{ij}^{\tau}})^2$ up to time step τ and ε is a small constant to avoid division by zero.

Adadelta

Adadelta [192] is an extension of Adagrad developed to overcome the decay of learning rates throughout training. Instead of accumulating all the previous squared gradients, the idea is to restrict the accumulation over a fixed size time window. This is efficiently performed by calculating a decaying average of the previous squared gradients. The accumulation v_{ij} in Eq. A.36 becomes:

$$v_{ij}^{\tau} = \gamma v_{ij}^{\tau-1} + (1-\gamma) \left(\frac{\partial E^{\tau}}{\partial w_{ij}^{\tau}} \right)^2 \tag{A.37}$$

where γ controls the exponential decay rate of the average v_{ij}.

Adam

The Adaptive Moment estimation (Adam) [193] is yet another variant which accumulates the decaying average of squared gradients like Adadelta as well as a decaying average of gradients. The former (m_{ij}^{τ}) estimates the first moment (mean) of past gradients, while the latter (v_{ij}^{τ}) is an estimate of the second raw moment (uncentred variance). These estimates are computed as follows:

$$\begin{aligned} m_{ij}^{\tau} &= \gamma_1 m_{ij}^{\tau-1} + (1-\gamma_1)\frac{\partial E^{\tau}}{\partial w_{ij}^{\tau}} \\ v_{ij}^{\tau} &= \gamma_2 v_{ij}^{\tau-1} + (1-\gamma_2)\left(\frac{\partial E^{\tau}}{\partial w_{ij}^{\tau}}\right)^2 \end{aligned} \tag{A.38}$$

where γ_1 and γ_2 are hyperparameters to control the decay rates. These moment estimates are then corrected to avoid an initial bias towards zero as follows[1]:

$$\begin{aligned} \hat{m}_{ij}^{\tau} &= \frac{m_{ij}^{\tau}}{1-(\gamma_1)^{\tau}} \\ \hat{v}_{ij}^{\tau} &= \frac{v_{ij}^{\tau}}{1-(\gamma_2)^{\tau}} \end{aligned} \tag{A.39}$$

The Adam update rule is finally computed as:

$$w_{ij}^{\tau+1} = w_{ij}^{\tau} - \frac{\eta}{\sqrt{\hat{v}_{ij}^{\tau}+\varepsilon}}\hat{m}_{ij}^{\tau} \tag{A.40}$$

The choice of the best optimiser (SGD, momentum, Adagrad, Adadelta, Adam and others) depends, among others, on the application, the network architecture, the training data and on a trade-off between speed and performance. A fast convergence of deep networks can generally be obtained with adaptive learning rate methods (e.g. Adagrad, Adadelta, and Adam), sometimes at the cost of lower accuracy.

Weights initialisation

The initialisation of the parameters (weights and biases) to start the learning process has not been discussed so far, yet it is a key step which requires careful design. While a first guess might be to initialise all the parameters to zero, it does not work in practice as they all compute the same gradients and do not allow gradient descent learning. Indeed, asymmetry is required in the weights initialisation. Also, one must pay attention to the effect of "vanishing" or "exploding" gradients that may occur

[1]Note that in Eq. A.39, $(\gamma_1)^{\tau}$ represents "γ_1 to the power of τ" as opposed to m_{ij}^{τ} which is the value of m_{ij} at time step τ.

generally in deep architectures with respectively too small or too large initial weights (see Section A.3.2).

A common initialisation method is the Gaussian initialisation. With this method, the weights are drawn from a Gaussian distribution of zero mean and usually small variance (e.g. 0.01). A more recent and frequently used method is the Xavier initialisation [181]. With the Xavier approach, the initialisations are also drawn from Gaussian distributions. However, the variance is a function of the number of input and output connections.

$$var(w) = \frac{2}{n_{in} + n_{out}} \tag{A.41}$$

where n_{in} and n_{out} are the number of input and output connections respectively. In the fully connected networks presented in this section, a weight w^l considers $n_{in} = m_{l-1}$ and $n_{out} = m_l$ numbers of connections. This initialisation keeps the initial parameter values in a certain range to avoid the signal to shrink or grow extensively. Note that it is common to initialise the biases to zero as the symmetry is broken by the weights initialisation.

A.2.4 Regularisation methods

Overfitting is a frequent problem in machine learning when training a model. It occurs when an excessively complex model describes the noise in the training data. The overfitted model will often not generalise well to unknown test data. An example of 1D overfitting is illustrated in Figure A.8. Neural networks are particularly prone to overfitting due to the large number of parameters to train as extremely complex models can be learned to fit the training data. Several regularisation methods are used to prevent overfitting by generally penalising the complexity. Commonly used regularisation methods include weight decay, early stopping, dropout, and artificial data expansion.

Weight decay

The weight decay is a regularisation method that penalises large weights [194]. For batch learning, Eq. A.33 becomes:

$$\Delta w_{ij} = -\eta \left(\frac{\partial E^{(k)}}{\partial w_{ij}} + \lambda w_{ij} \right) \tag{A.42}$$

where λ is the weight decay hyperparameter. Note that Eq. A.42 originates from a penalty added to the error function:

$$\tilde{E} = E + \frac{\lambda}{2} \sum_i w_i^2 \tag{A.43}$$

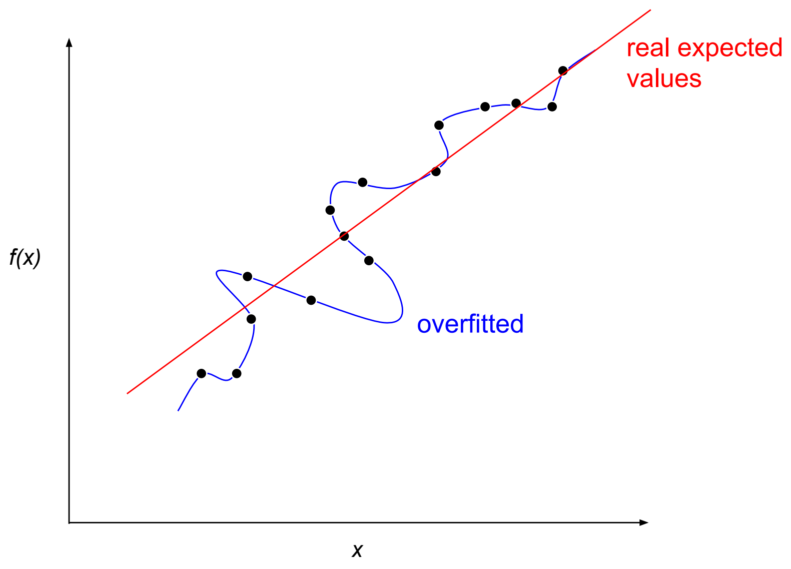


Figure A.8: An example of overfit (blue curve).

where w_i represent all the weights and biases of the network. By penalising large weights, the curvatures of the mapping function are also reduced, thus reducing the model flexibility to overfit the training data.

Early stopping

Early stopping is another regularisation method to limit the complexity of a network. The idea is to use an independent validation set during training to evaluate the performance of the network on unknown data. When the error on the validation set starts saturating and increasing, the training is stopped to avoid overfitting. An example of early stopping is illustrated in Figure A.9.

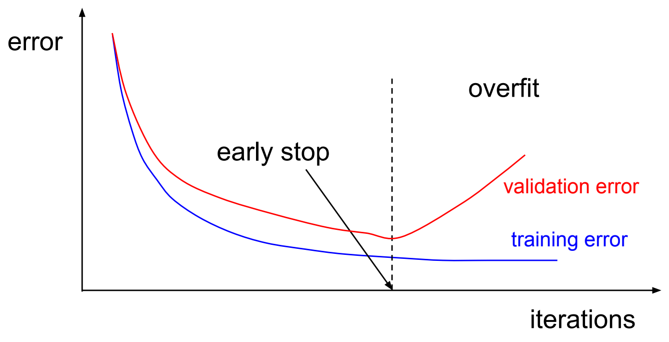


Figure A.9: An illustration of overfitting the training data and early stopping method by evaluation of the model on an unknown validation set during training.

Dropout

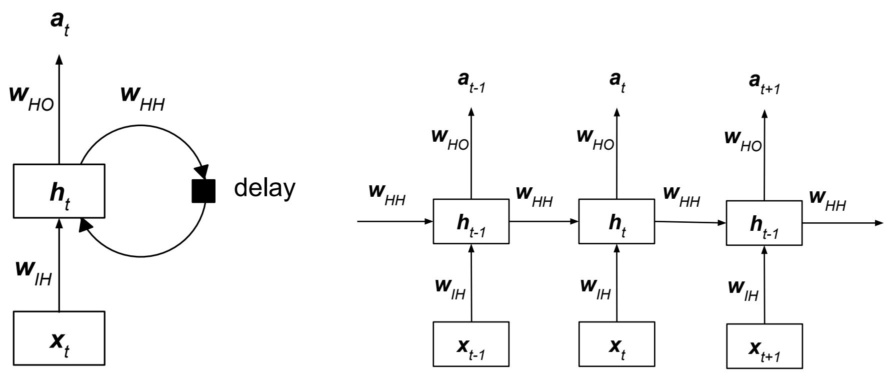
Dropout is a recent regularisation method [195] which refers to the “drop out” of certain neurons during training. The effect of dropout is similar to averaging the predictions of multiple independently trained models while being considerably more efficient. At each training iteration (forward and backward passes), individual neurons and their connections are “dropped out” of the network with a probability of $1 - p$. Thus, multiple reduced networks are trained, preventing complex co-adaptations of the parameters by learning various independent representations of the training data. At test time, the entire network is used without dropout, behaving like an ensemble model. The neuron outputs are weighted by a factor p to keep the same expected values as in the training phase. On top of the efficient regularisation, dropout also reduces the training time.

Artificial data expansion

This regularisation method mainly relies on obtaining invariances to transformations of the input data that may occur in unknown test data. A typical example is a neural network with images as inputs for an object recognition task. The network should recognise objects of varying scale, rotation, or position in the image as the same object. However, the training set might provide only a limited number of such variations. A simple way of avoiding the network to overfit certain orientations, scales, and positions present in the training data is to artificially expand this data with controlled transformations. A number of random rotations, scalings, and croppings can be applied to the training images to expand it and explicitly teach the network to recognise such variations.

A.2.5 Recurrent neural networks

All the architectures presented so far are feedforward in the sense that the connections between neurons do not form a cycle or a loop. These feedforward networks make the assumption that the inputs are independent of each other. Recurrent Neural Networks (RNN) [1], on the other hand, make use of the inter-dependency of sequential data. RNNs contain at least one feedback connection and thus form a directed cycle. They were developed for sequential data as the inner loops create internal states of the network used to analyse sequential (generally temporal) behaviours. The fully recurrent network is a basic RNN architecture which is similar to an MLP with the hidden activations fed back into the input of that same hidden neuron at the next iteration as shown in Figure A.10a. Note that in this illustration, the input vectors and hidden neurons are represented as a single rectangle to illustrate the recurrence in a graph instead of a neurons representation. This differs from other



(a) feedback loop **(b)** unfolded graph

Figure A.10: A graph representing a basic fully recurrent network. Note that the inputs, weights, hidden states, and outputs are vectors. (a) graph with a loop feeding the previous hidden state back into the network, (b) unfolded graph over time.

figures in the appendix and is specified by the bold fonts in which $\mathbf{h}_t$ represents the hidden states of the neurons. Figure A.10b shows the same network unfolded over time. The weights $\mathbf{w}_{IH}$, $\mathbf{w}_{HH}$ and $\mathbf{w}_{HO}$ connect respectively the input to the hidden neurons, the hidden neurons to the hidden neurons at the next iteration and the hidden neurons to the output. Note that these weights are shared across all sequential steps. Backpropagation can be used to train this network although it is often referred to as Back Propagation Through Time (BPTT) as the error for a given sequence is the sum of the errors at each iteration.

$$E_{seq} = \sum_t E_t \tag{A.44}$$

where t spans the iterations of the sequence. In this way, the update of the weights through gradient descent depends on the activation and error at previous iterations.

$$\Delta w_{ij} = -\eta \sum_t \frac{\partial E_t}{\partial w_{ij}} \tag{A.45}$$

A.2.6 Unsupervised learning

The network architectures introduced so far are trained in a supervised manner. It is supervised in the sense that the network learns a function from labelled training data. Each training sample is fed into the network together with the desired output value which can be a class for a classification problem or a real value in a regression task. In other words, an input sample is presented to the network, specifying what is being shown in order to teach the network how to recognise it. This is how the error between the prediction and the target was calculated in the previous section for backpropagation. Ideally, the network trained on this labelled data will be able to generalise, i.e. perform the learned task on unknown data.

Unsupervised learning, on the contrary, aims at building representations that describe the structure of unlabelled input data. The backpropagation method as described in Section A.2.3 cannot be implemented as such as the error cannot be calculated between the prediction of the network and the desired target value. Most popular unsupervised machine learning methods include the K-means clustering algorithm and the Gaussian Mixture Models (GMMs). Unsupervised learning has also been widely investigated in ANNs and several approaches have been developed including Hebbian learning and other unsupervised neural networks described below.

Hebbian learning

Hebbian theory [196] is based on the idea that neurons that activate together wire together. Thus, in a Hebbian learning scheme, the weight between two neurons will increase if they activate at the same time and will reduce if they activate separately. An on-line formulation of the Hebbian learning can be expressed as:

$$w_{ij}^{\tau+1} = w_{ij}^{\tau} + \eta x_i^{\tau} a_j^{\tau} \tag{A.46}$$

where η is still the learning rate, $w_{ij}^{\tau+1}$ is the updated weight connecting the i^{th} input x_i^{τ} to the neuron with activation a_j^{τ}. This formulation is highly unstable due to unbounded variations of the weights and is not often used as such in ANN unsupervised learning. Other methods based on this approach have overcome the instability and improved the learning process including BCM, Oja's learning, and the generalised Hebbian method.

Unsupervised neural network

An Auto-Encoder (AE) is an unsupervised ANN which learns a representation of an input dataset to extract meaningful features from the data. AEs use an input layer, at least one hidden layer and an output layer to encode and decode (reconstruct) the input data. A basic example of AE is illustrated in Figure A.11. The learning of AEs is performed by comparing the reconstructed input $\mathbf{x}'$ to the real input $\mathbf{x}$ and adapting the weights through backpropagation to minimise the error in the reconstruction. The latent representation $\mathbf{z}$ learned in the intermediate layers of AEs have been successfully used for dimensionality reduction, feature learning, classification and as a generative model (variational AE [197]).

Another simple shallow example of unsupervised network is the Restricted Boltzmann Machine (RBM) [198]. An RBM architecture is very similar to an AE in the sense that it learns to encode and decode (reconstruct) input data. RBMs,

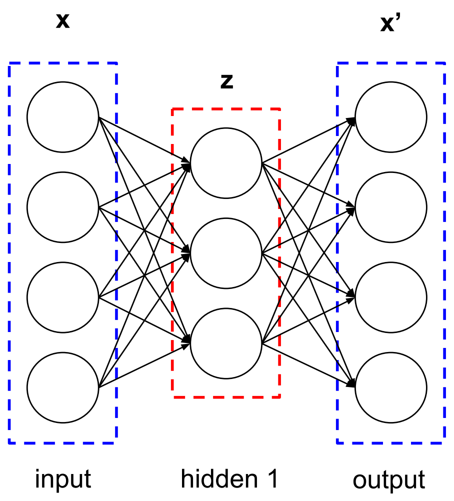


Figure A.11: The architecture of a simple AE.

however, contain only two layers as shown in Figure A.12. The training of an RBM is also different from AEs. The error between the reconstructed and original inputs is calculated by contrastive divergence. The learning is then carried by a gradient-based optimisation comparable to the backpropagation explained in Section A.2.3 to obtain a reconstruction of the input as close as possible to the original input. An RBM is a generative model, as opposed to discriminative models, in the sense that the network generates a model of the data by learning a joint probability distribution of the inputs **x** and activations **a**. This joint probability arises from the reconstruction (backward pass) with the same weights as the forward pass. Note that while a classic AE can represent the input data in a lower space, it is not a generative model. RBMs can be stacked into multiple layers to form a Deep Belief Network which will be introduced in Section A.3. Note that in Figure A.12, the weights and biases are the same in both the forward and backward passes and the biases are not represented for simplicity. The RBM is *restricted* as opposed to a standard Boltzmann Machine (BM) as there is no connection between neurons of the same layer.

A.2.7 Reinforcement learning

Supervised learning was introduced in the previous sections, where target labels (ground truth) are provided together with training samples to teach the models. Unsupervised learning was also described in the previous section, where models learn to describe hidden structures in training data without training labels. For completeness, Reinforcement Learning (RL) is now briefly described. RL can be defined as a learning method between supervised and unsupervised as the model

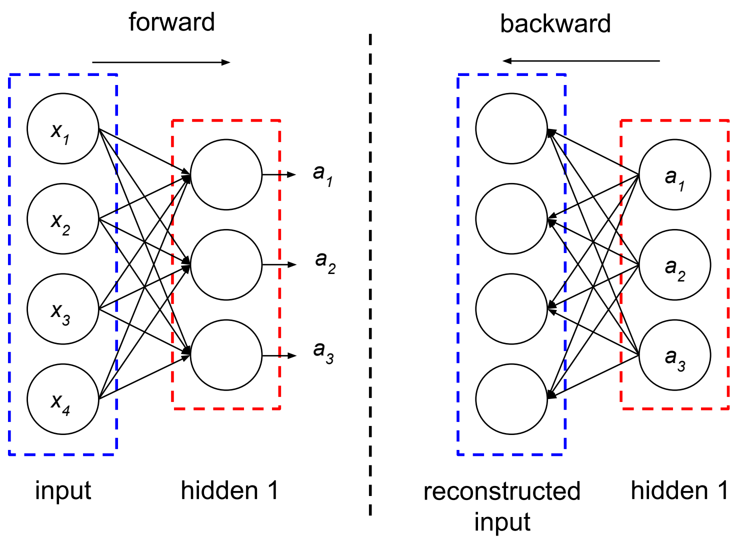


Figure A.12: The architecture of a Restricted Boltzmann Machine with three input and four hidden neurons including forward (left) and backward passes (right).

makes a prediction and is taught whether it is correct or not with sparse and time-delayed labels. The model referred to as *agent* interacts with the *environment* by trying different actions and is trained by getting feedback in the form of *rewards* from the consequences of these actions. RL in combination with deep neural network (Section A.3) is often used in robotics or to teach a model how to play games such as Atari [199] or Go [200]. This learning approach is extensively studied as it shares ideas with the way humans learn and enables models to learn directly from the results of their actions on the environment; yet, RL is outside the scope of this thesis.

A.3 Deep learning

So far, the described networks have not been referred to as deep neural networks. A deep neural network is simply an ANN with many layers. Particular attention is required to train these more complex networks and this is what deep learning does. In particular, problems such as the “vanishing gradients” and the “covariate shift” often arise from training networks with many layers using classic gradient-based learning methods described previously. For several decades after the development of the first neural network methods, researchers were only able to train shallow networks with three or fewer layers. It was only in 2006 that advances in deep learning and the computational power of computers allowed deeper architectures [198, 201, 202] to be trained. This section introduces some difficulties that arise with training deep models as well as key methods to overcome them. Finally, some of the major deep network architectures are presented.

A.3.1 Regularisation

Deep architectures are prone to overfitting and the methods introduced in Section A.2.4, including Dropout, weight decay and data expansion are commonly used in deep learning. Also, very large datasets are often needed to generalise better to unknown data. For instance, the ImageNet 2012 training dataset [50], used in many computer vision deep learning methods, contains more than one million images grouped into one thousand classes.

A.3.2 Vanishing gradients

The gradients in a deep learning scheme are unstable as they are generally the product of many other gradients from deeper layers. The gradient values tend to decrease dramatically, or sometimes explode, as they are backpropagated towards the input layer. This is a frequent problem which arises with training deep architectures referred to as vanishing/exploding gradients. This matter makes the training of the parameters in early layers difficult as the updates become extremely slow and likely to get caught in a local minimum or to degenerate. Several methods have been developed to avoid vanishing gradients.

The choice of the activation function largely influences the vanishing gradients problem. ReLU is generally preferred over other activation function (e.g. sigmoid and tanh) because it is not saturated. Therefore, a ReLU function does not squash the values into a small range as the other activation functions do ($[0,1]$ or $[-1,1]$). Mapping the input space into such a narrow space means that even large variations in the input space result in small variations in the squashed feature space, which causes small gradients. The vanishing problem can also be explained with the derivatives of the sigmoid and tanh functions which lie mostly in a range close to zero. Thus, the product of the partial derivatives in the chain rule (Eq. A.17 and A.18) results in very small gradients in early layers. The cascade of many layers accentuates this phenomenon which is frequently encountered in deep learning with squashing activation functions. Note that alternatives to ReLU are sometimes used such as the "leaky" ReLU which avoids the zero gradients for negative input values.

A small learning rate, together with an appropriate initialisation of the weights (when trained from scratch), helps to obtain an initial convergence and stable gradients. The initialisation of the weights is highly related to the vanishing and exploding activations which in turn also affect the gradients and the learning process. Briefly, if the initial weights are too small, the variance may dramatically decrease in the feedforward pass. If they are too big, the activations in the higher layers may saturate (with a sigmoid or tanh function) or explode with the unbounded ReLU. This is the reason why the Xavier initialisation A.2.3 is normalised to keep the weights in a reasonable range and avoid vanishing and exploding activations.

An early attempt to control the range of values of the activations throughout the network was to normalise the activations of intermediate layers. This Local Response Normalisation (LRN) method [15] has been outperformed by the recent batch normalisation method [2] which will be introduced shortly.

A.3.3 Internal covariate shift

Another frequent problem with deep learning is the "covariate shift". The covariate shift refers to the change in the input distribution of the intermediate layers of a network. The inputs to intermediate layers are affected by the parameters of all previous layers. Therefore, small changes to the weights are amplified throughout the network and result in large changes in the input distribution of intermediate layers. This problem is naturally amplified with deep architectures. A solution to the covariate shift is the use of batch normalisation.

A.3.4 Batch normalisation

Batch Normalisation (BN) [2] is a recent method to help training deep neural networks. The idea is to normalise, scale and shift the inputs of the activation functions $h^l(\mathbf{x})$. This normalisation regularises the network and reduces the dependence of the gradients on the range of its weights and their initialisation. It largely helps the vanishing gradients and allows the gradient descent algorithm to use higher learning rates while ensuring the convergence. Moreover, BN reduces the covariate shift of the input of intermediate layers. In turn, BN is generally preferred to most regularisation methods and other methods for avoiding vanishing gradients and internal covariate shift in deep architectures.

A.3.5 Deep recurrent networks

Deep recurrent network architectures have been extensively researched for speech, handwriting, and video analysis. Deep recurrent networks are merely RNNs (see Section A.2.5) with many layers. With long input sequences, the covariate shift and vanishing gradients are severe in RNNs and require extra attention. This is due to the fact that the chain rule used to calculate the partial derivative of the error at time t in Eq. A.45 involves many partial derivatives from the previous steps. A deep RNN cannot remember everything from the past and a method is required to retain useful information. To this end, the Long Short Term Memory (LSTM) [203] is often used in deep RNNs. This approach prevents the gradients from vanishing or exploding during backpropagation through time by memorising the error values. The main idea of LSTM is to include *gating* units in the recurrence which will decide when to write into memory, read from it or erase it. Very deep LSTM networks can therefore learn from long sequences which require a selective memory of long sequential events. A basic LSTM block is depicted in Figure A.13 which replaces the simple hidden layer in the classic fully recurrent network shown in Figure A.10b. The gates are sigmoid

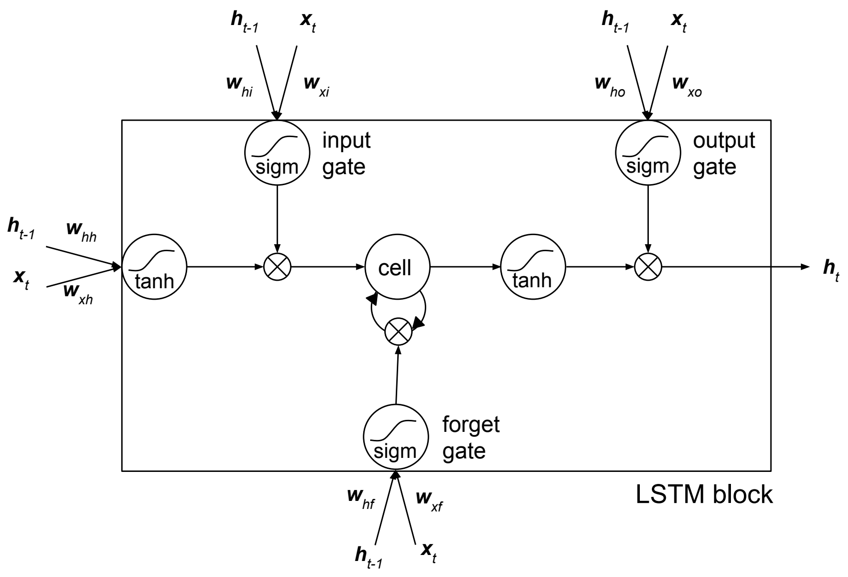


Figure A.13: A basic LSTM block.

functions which behave like smooth switches and output approximately one or zero based on the current input $\mathbf{x}_t$, the previous hidden state $\mathbf{h}_{t-1}$ and trainable weights. In turn, the output of a gated unit may allow the signal (input, cell memory or output) to propagate or block it as represented by the multiplication signs which perform a Hadamard product. The *input* gate controls whether the current input should be written into memory (cell) and computes its output as:

$$i_t = sigm(\mathbf{w}_{xi}\mathbf{x}_t + \mathbf{w}_{hi}\mathbf{h}_{t-1} + b_i) \tag{A.47}$$

The *forget* gate decides when to erase the memory (with another recurrence loop). Finally, the *output* gate controls when to read from the memory. The forget and output gates compute their output similarly to the input gate with different weights.

Note that the LSTM block in Figure A.13 and the simple recurrent network in Figure A.10 in which the LSTM can replace the hidden layer, have many variants including the feedback loops, activation functions (usually hyperbolic tangent and sigmoid), the number of hidden layers and of LSTM blocks, as well as peepholes [204] which enable the gates to “look at” the cell state.

A.3.6 Deep unsupervised methods

Shallow unsupervised networks introduced in Section A.2.6 have been extended, as most neural networks, to deep architectures. Some deep unsupervised neural networks are introduced in the following sections.

Deep belief network

A Deep Belief Network (DBN) [201] is a deep feedforward unsupervised network which consists of stacked RBMs. This network can be used to pre-train a deep network. The pre-trained weights can be used as initial weights to a backpropagation training in a supervised manner. It can be useful for instance to make use of a large amount of unlabelled data together with a small labelled dataset. A DBN is trained layer by layer. The first RBM is trained by reconstructing the inputs as shown in Section A.2.6. The hidden outputs of the first RBM are used as inputs to the second RBM and so on until the entire network is trained.

Stacked Auto-Encoder

A stacked Auto-Encoder is a deep feedforward network obtained by stacking multiple AEs. Similarly to a DBN, the first AE is trained then its outputs are used as inputs to the following AE. A frequently used alternative approach is the stacked denoising Auto-Encoder. This is used to avoid inferring the identity mapping through a classic stacked AE training and to learn meaningful representations of the data. In a stacked denoising AE training, the input is corrupted by adding noise and the network is trained to output an uncorrupted version of the input data. The error is simply calculated between the reconstructed output and the original non-corrupted input. A trained stacked AE or stacked denoising AE can be used, like a DBN, for pre-training a network which will be finetuned on a supervised task.

Generative adversarial networks

A Generative Adversarial Network (GAN) [9] involves the training of two models which compete against each other, a generative and a discriminative model. A discriminative model maps the input data $\mathbf{x}$ to the desired class labels $\mathbf{y}$ in a classification or regression scheme, i.e. it learns the conditional probability distribution $p(\mathbf{y}|\mathbf{x})$. A generative model (e.g. RBM Section A.2.6, DBN Section A.3.6, deconvolutional network [205]) learns the joint probability distribution of the input data and labels. A conditional probability can then be formed from the joint probability with the Bayes' rule.

In a GAN, a generative network G is trained to capture the data distribution and generate samples that seem to come from the training data. A discriminative network

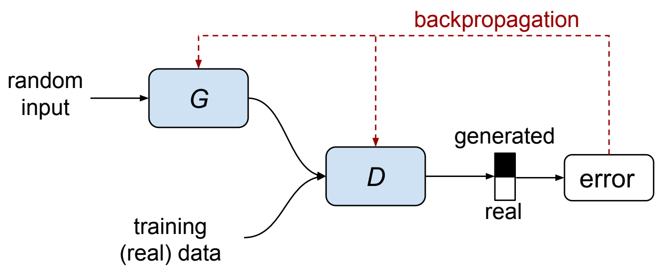


Figure A.14: An illustration of the GAN structure. The generative model *G* is trained to generate samples that seem to originate from the real training data (i.e. maximise the discriminator's error), while the discriminative model *D* is trained to discriminate the generated samples from the training data (i.e. minimise the error).

D, on the other hand, is trained to discriminate samples created by *G* from those in the training data (binary classification optimised with backpropagation). Figure A.14 illustrates a simple GAN structure. To produce a fake sample, *G* is fed with an input from a fixed random distribution and maps it to a data distribution which is optimised through training to approximate the distribution of the training data. This optimisation is carried by a backpropagation algorithm to minimise the accuracy of *D*, i.e. maximise its error.

A.4 Convolutional neural networks

CNNs are a type of ANN for analysing grid-like data including 1D time-series (e.g. sound), 2D images, 3D volume data, and videos. This section provides a brief overview of CNNs including the history, the description of different layers and concepts involved in the current architectures and applications.

A.4.1 Overview

A CNN is a supervised feedforward ANN (although unsupervised and recurrent variations have been developed). It is, like other networks introduced in the previous section, a succession of linear and non-linear operations applied to the input data. The analysis of the visual cortex [206] inspired the arrangement of neurons which respond to overlapped regions tiling the input grid-like data. In this section, the input data is an image (i.e. 2D array of pixels) unless stated otherwise. The neurons in a CNN behave as a bank of filters with local receptive fields in the input image and enable to exploit the local correlation present in natural images (i.e. correlation of pixel values in image neighbourhoods) in a deep learning approach. The key component of a CNN is the convolution layer which consists of a set of small trainable filters

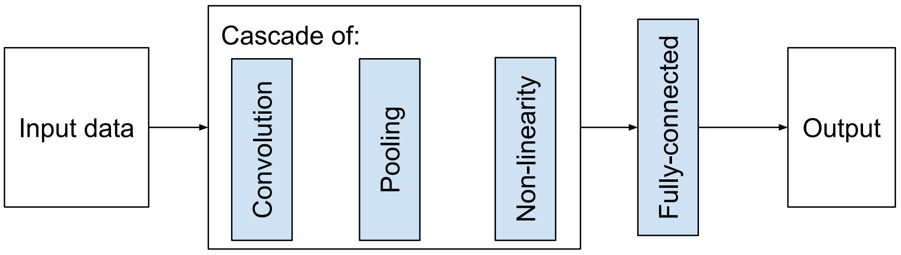


Figure A.15: A basic overview of a Convolutional Neural Network architecture.

locally connected to the output of the previous layer. A basic CNN approach can be summarised by a succession of convolution, pooling, non-linear and fully-connected layers as shown in Figure A.15. Each layer will be explained in detail in the next sections. CNNs are generally used for classification or regression and are trained in a supervised manner. The convolution filters and fully connected weights are trained by backpropagation and SGD (see Section A.2.3). Once successfully trained, the filters at multiple layers of a convolutional network respond to features of different complexity similarly to the visual cortex [206]. The bottom layer has been shown to detect edge-like features (similar to a Gabor filter set) while deeper layers detect more complex and abstract features with larger receptive fields (see Section A.4.7). This complexity in deep layers arises from the successive combination of simpler features using linear and non-linear operations. The receptive field in a CNN is an important notion illustrated in Figure A.16. The receptive field of a neuron is the area that it covers in the input image by a cascade of connection. The size of the receptive field of a neuron in the first convolution layer is the kernel size. The receptive field of the neurons increases with the depth of the CNN through convolution and pooling layers. Note that pixels in the centre of the receptive field will have a greater impact on the output of a neuron due to the multiple connections. For the same reason, the gradients at a certain intermediate neuron will influence more the update of the weights connected to the pixels at the centre of its receptive field than those at the border [207].

A.4.2 Brief history

Classic neural networks such as MLPs introduced in Section A.2.2 do not scale well to image analysis. The number of pixels in an image is large and a fully-connected approach leads to a very large number of weights which are difficult to optimise and prone to overfitting.

The discovery of cells in the visual cortex responding to sub-regions of the visual field was made in 1968 [206]. It was shown that simple cells respond to edge-like

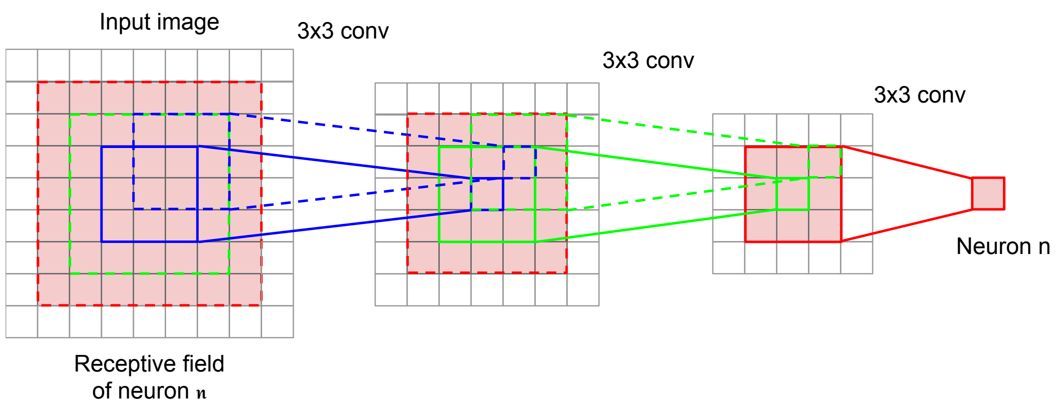


Figure A.16: An illustration of the receptive field of a neuron after three convolution layers. The receptive field of the neuron n is the red area in the input image connected to this neuron through the convolutions. Best viewed in colour.

patterns while more complex ones detect larger, more global and more invariant features. Based on this work, research on convolutional networks started in the 1980's with the predecessor of the CNN named "neocognitron" [4]. This method already used multiple layers (two hidden layers) to gradually combine simple local features into more complex patterns. Several key concepts of CNNs were developed in this work including the local connectivity, the weight sharing and a type of subsampling (averaging pooling). The neocognitron was not trained with backpropagation and the first CNN trained with backpropagation was introduced in 1989 [208]. In 1998, an improved version of CNN named LeNet5 was developed [5] for handwriting recognition which resembles many recent networks in terms of operations (e.g. convolution, non-linear activation, max-pooling, and fully-connected layers) applied to the data but not in terms of depth and complexity. Except for the handwriting application, CNNs had limited attention in the computer vision community at this time. Their success was mainly restricted by the computational power and memory of the computers, the lack of labelled data, and the difficulties to train deep architectures. The year 2006 was important for deep learning and CNNs. The work of Hinton et. al on DBNs [201] and on deep AEs [198] attracted interest for deeper architectures and allowed pre-training complex deeper networks. The parallel computing of GPUs significantly improved the speed and memory capacity of the computers, making the training of deeper networks possible. More powerful CNN architectures were developed at the same time [202, 209], including a backpropagation training of a max-pooling CNN, and obtained the state of the art in several computer vision tasks.

Until 2010, CNNs achieved excellent results on small images and simple recognition benchmarks which required limited abstraction of concepts and invariance. However, they were outperformed by classic handwritten machine learning algo-

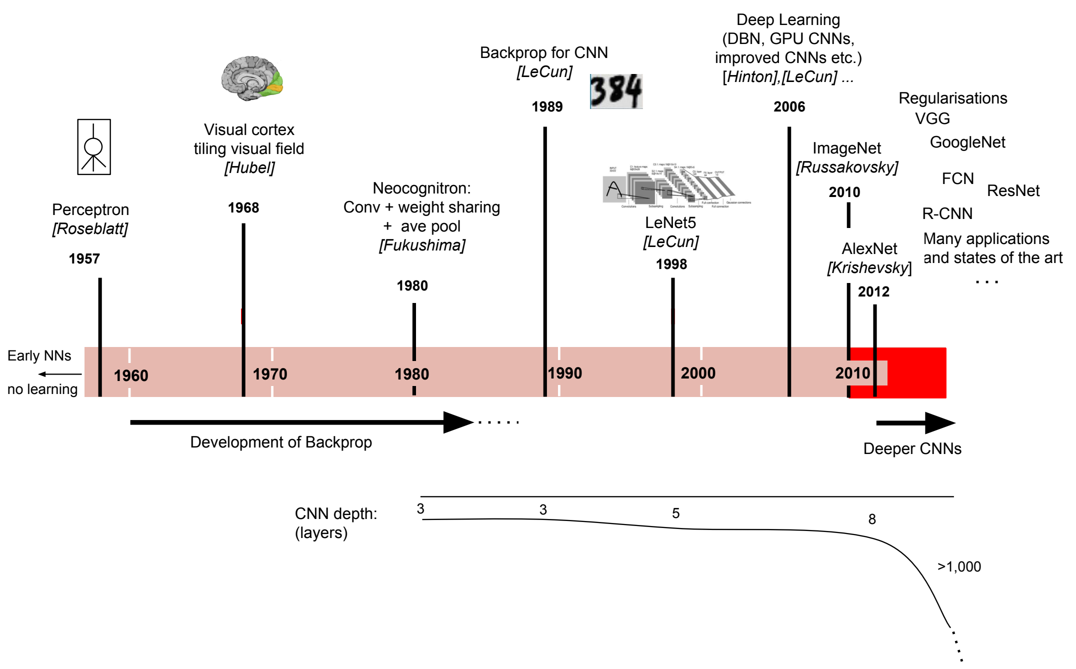


Figure A.17: A timeline of CNN history.

rithms on more complex problems with larger images and fewer training data. The second success of CNNs in the last seven years was highly related to ImageNet [50]. This large labelled dataset enabled training deeper architectures with less overfit of the training data. In particular, the seven layer AlexNet [15] trained on ImageNet marked an important advance in CNNs. An interesting aspect of the CNN is its domain transferability. Pre-training a deep network on a very large dataset such as ImageNet and finetuning it on another smaller dataset has shown that the learned features generalise well to other tasks. Note that in finetuning, the parameters in the top layer are optimised for the target task while the weights in earlier layers are generally either kept unchanged or only slightly modified. It resulted in a new state of the art even on datasets with a small amount of training data. Since 2012, CNN has become the most popular approach in computer vision and other fields, establishing a rapidly changing new state of the art on most of the benchmarks. In particular, many new regularisation methods, learning algorithms, architectures, and applications have been proposed. The number of layers used in a network has increased exponentially in the last years with architectures such as VGG [18, 19] (7, 16 and 19 layers), GoogleNet [164] (22 layers) and deep residual networks [210] (50, 101, 152 layers and even >1,000). These architectures will be covered in more details in Section A.4.5. In the meantime, significant work has been conducted to attempt to visualise and understand how and what CNNs learn [211] as described in Section A.4.7.

A.4.3 Main building blocks

The various building blocks of a CNN will now be introduced. The reader should keep in mind that a network is built with a succession of these layers and that many different architectures exist. The same notations as in Section A.2 are kept throughout this section.

Convolution

The convolution layer is the most important building block of a CNN. It incorporates several key concepts which are well suited to the analysis of images and other grid-like data. The convolution of two functions h and w is given by:

$$(h*w)(t) = \int_{-\infty}^{\infty} h(\tau)w(t-\tau)d\tau \tag{A.48}$$

where h and w are defined in $\mathbb{R}$.
For a discrete variable x, the convolution operation is:

$$(h*w)(x) = \sum_{n=-\infty}^{\infty} h(n)w(x-n) \tag{A.49}$$

In two dimensions (image), the discrete convolution becomes:

$$(h*w)(x,y) = \sum_{n_x=-\infty}^{\infty}\sum_{n_y=-\infty}^{\infty} h(n_x,n_y)w(x-n_x,y-x_y) \tag{A.50}$$

In CNNs, functions operate on multi-dimensional arrays of data. In the general case (2D images), the dimensionality of input and output data is the product of three dimensions $C \times H \times W$ for respectively the number of channels, the height, and the width. Note that the number of channels in the input image is generally three for colour or one for greyscale data. The number of images per batch (see batch training in Section A.2.3) is sometimes included in the data dimension as multiple images are simultaneously fed. The inputs of a convolution layer are called input channels while the outputs are referred to as feature maps after activation (see non-linear activation in Section A.2.2). A convolution layer is mainly defined by a set of filters (or kernels) by which the input data is convolved in the forward pass. Thus, the main parameters of a convolution layer include the number and size (usually small) of the filters. An output feature map is obtained by summing the convolution results of each input channel with the appropriate kernel as shown in Figure A.18 and applying non-linearity (not represented in Figure A.18). The biases are not represented in the

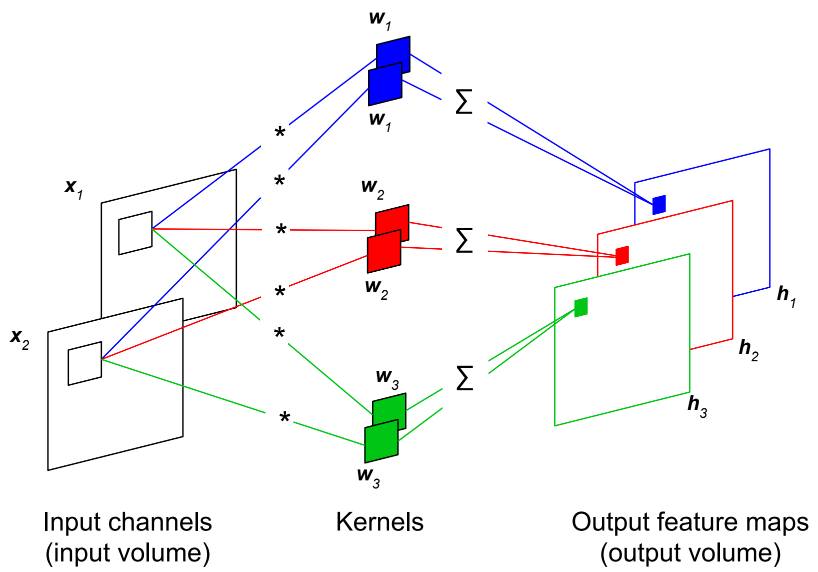


Figure A.18: A convolution layer with two input channels and three output feature maps. The activation functions are not represented for simplicity.

figure for simplicity. An output h_j is obtained as:

$$\mathbf{h}_j = \sum_{i=1}^{C} \mathbf{x}_i * \mathbf{w}_{ij} \tag{A.51}$$

where $\mathbf{h}_j$ is the j^{th} output of the convolution layer and $\mathbf{w}_{ij}$ is the kernel between the i^{th} input channel and the j^{th} output. The sum runs over the C input channels. The operator $*$ is a 2D convolution as defined in Eq. A.50. The filters scan the input channels with a certain stride by which they slide. A large stride results in smaller output feature maps. The input channels can be zero padded with a desired size to convolve the edges of the input channels. For instance, a padding of one pixel can be used to convolve an image with a 3×3 kernel, maintaining the output size equal to the input size. The stride, padding, and kernel size are illustrated in Figure A.19.

Local connectivity (or sparse connectivity) is a key concept of the convolution layer. Unlike fully connected architectures (see MLPs Section A.2.2), convolution layers ensure that each neuron of the network is connected to a small number of neurons in the input channel equal to the size of the filters as shown in Figure A.18 and A.19. On top of capturing local dependencies, the local connectivity also performs a large reduction of trainable parameters. Indeed, the high dimensionality of the input images (number of pixels) makes the fully-connected approach very impractical.

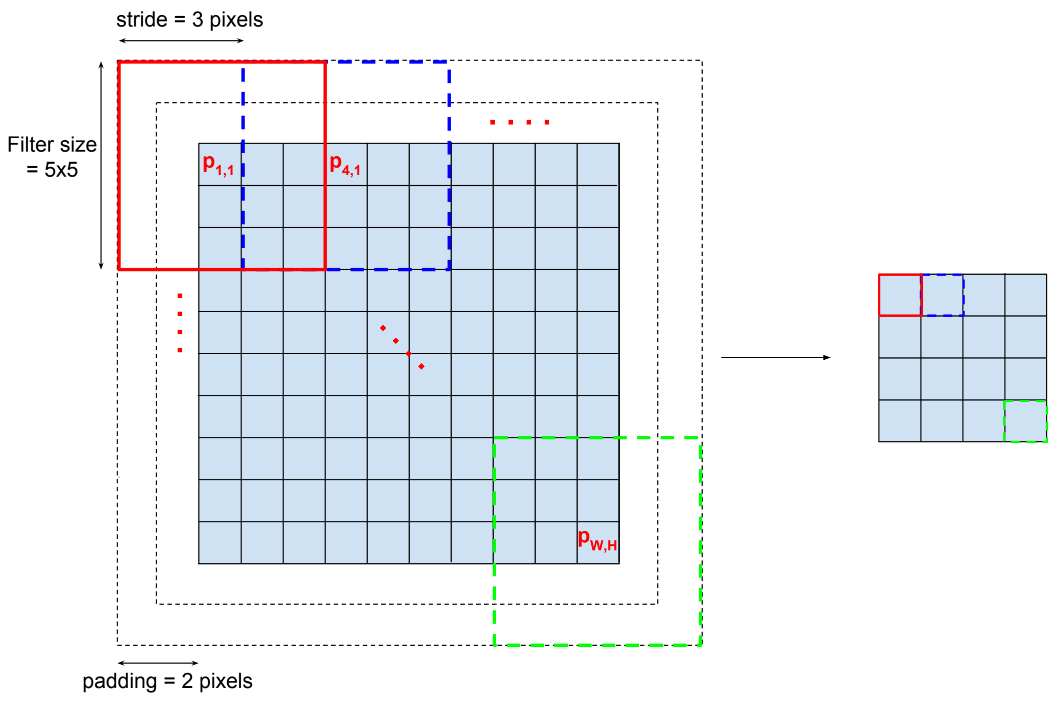


Figure A.19: An illustration of the filter size, stride and zero padding in the forward pass of a convolution layer. W and H are respectively the width and height of the input channel.

The second key concept of the convolution layer is the weight sharing. In a CNN, the hidden neurons are grouped into feature maps. All the hidden neurons within a feature map share the same parameters (i.e. filters) and each hidden neuron within a feature map covers a particular local part of the input channels. This can be interpreted in the sense that the same features (edges, corners, and more complex patterns) are sought everywhere across the image. This is particularly true in the analysis of textures with repetitive patterns. Note that for certain types of images, this might not be the case and one might allow different filters for different parts of the image. For a face analysis task, for instance, different filters can be used for different parts of the image to detect the mouth, the eyes etc. [184].

To summarise, the hyperparameters of a convolution layer include the number of filters (or number of output channels), the filter size, the initialisation of the filters (see Section A.2.3), the stride and the padding size. The size (height and width) of the output feature maps is a function of the input size of the convolution layer and of the hyperparameters. It can be computed as:

$$l_{out} = \frac{(l_{in} - l_w + 2 \times pad)}{s} + 1 \tag{A.52}$$

where l_{out} and l_{in} are the size (height or width) of the input channels and of the output feature maps respectively. l_w is the filter size, *pad* is the padding size, and s is the stride. Note that the number of trainable parameters (weights and biases) in a convolution layer can be calculated as $N_i \times N_o \times l_w^2 + N_i$, where N_i and N_o are the number of input channels and number of output feature maps respectively k is the kernel height and width.

The backpropagation and SGD used to train a CNN are similar to the MLP (Section A.2.3), keeping in mind the local connectivity and weight sharing. In particular, the gradients at a convolution layer w.r.t. the weights are backpropagated using:

$$\begin{aligned}
\frac{\partial E}{\mathbf{w}_{ij}^{l}(n_x,n_y)} &= \sum_{x,y} \frac{\partial E}{\partial \mathbf{h}_j^{l+1}(x,y)} \frac{\partial \mathbf{h}_j^{l+1}(x,y)}{\partial \mathbf{w}_{ij}^{l}(n_x,n_y)} \\
&= \sum_{x,y} \frac{\partial E}{\partial \mathbf{h}_j^{l+1}(x,y)} \mathbf{x}_i^{l}(x-n_x, y-n_y) \\
&= \frac{\partial E}{\partial \mathbf{h}_j^{l+1}(x,y)} * \mathbf{x}_i^{l}(-n_x,-n_y)
\end{aligned} \tag{A.53}$$

Similarly, the gradients w.r.t. the inputs $\mathbf{x}_i^l$ for further backpropagation through earlier layers are computed as:

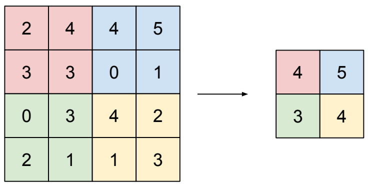
$$\begin{aligned}
\frac{\partial E}{\mathbf{x}_{i}^{l}(n_x,n_y)} &= \sum_{j}\sum_{x,y} \frac{\partial E}{\partial \mathbf{h}_j^{l+1}(x,y)} \frac{\partial \mathbf{h}_j^{l+1}(x,y)}{\partial \mathbf{x}_{i}^{l}(n_x,n_y)} \\
&= \sum_{j}\sum_{x,y} \frac{\partial E}{\partial \mathbf{h}_j^{l+1}(x,y)} \mathbf{w}_{ij}^{l}(x-n_x, y-n_y) \\
&= \sum_{j} \frac{\partial E}{\partial \mathbf{h}_j^{l+1}(x,y)} * \mathbf{w}_{ij}^{l}(-n_x,-n_y)
\end{aligned} \tag{A.54}$$

Non-linearity

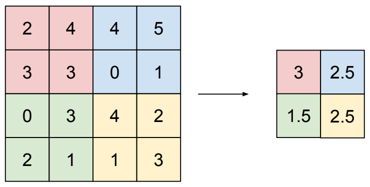
The reader should refer to Section A.2.1 for an explanation of the non-linearity in ANNs. ReLU layers are generally used after the convolution and fully-connected layers to activate the neurons as explained in Section A.2.1.

Pooling

The benefit of a pooling layer is three-fold. First, a pooling layer is used to reduce the size of the data throughout the network. Secondly, it helps the network to learn small transformation invariances (translation, scale, rotation) as the pooling operation detects features regardless its position in a small neighbourhood. Thirdly, it also increases the receptive field of the neurons. The receptive field is the area in the



(a) max



(b) average

Figure A.20: An example of a pooling layer (Forward pass) with 2×2 filters (a) max pooling, (b) average pooling.

input image to which a neuron is directly or indirectly connected as illustrated in Figure A.16. Receptive fields in the top layers are larger than in early layers due to the filter sizes and the stride of the convolution and pooling layers. The most popular pooling approach is "max" pooling. The max pooling layer outputs the maximum values of small non-overlapping patches in the input channels. A filter size, stride and padding size must be specified similarly to the convolution layer (Figure A.19). Note that, unlike a convolution layer, there is no trainable parameter in a pooling layer.

Another approach discussed in this thesis is "average" pooling. Figure A.20 illustrates an example of max and average pooling with a typical filter size 2×2. The same idea of splitting the input channels into non-overlapping patches is applied, and the average of each patch is calculated.

The gradients are backpropagated through a max pooling layer only through the neuron with maximum value (maximally activated in the forward pass) as changing non-maximum neurons does not affect the output. Therefore, the location of the maximum values in the input of a max-pooling layer during the forward pass must be saved for backpropagation. The gradients are copied to the maximum locations, the rest are set to zero. In an average pooling layer, all the input neurons covered by a pooling kernel are given the same gradient value, i.e. the output gradients divided by the kernel size. Average pooling can alternatively be seen as a special type of convolution with fixed weights.

Note that several recent approaches get rid of pooling operations [212] by using convolution layers with increased stride.

Fully-connected layer

Fully-connected layers are often used after a cascade of convolution, non-linear and pooling layers as shown in Figure A.15. A fully-connected layer is similar to an MLP layer, i.e. the neurons are connected to all the input neurons. The output of a fully-connected layer is a vector of dimension equal to the number of neurons. In a classification scheme, the last fully-connected layer contains a number of neurons equal to the number of classes. In a trained network, a high activation of the i^{th} output value of the last fully-connected layer reflects a high probability of the input image being of class i. Note that the fully-connected layer can be thought of as a 1×1 convolution layer. Several recent architectures replace the fully-connected layers by 1×1 convolution layers [6] to obtain a spatial map of output score vectors from input images of varying sizes.

Loss

The loss, as explained in Section A.2.3, measures the error between the output of a network and a target (or label) vector. The loss is used during training for the backpropagation of the gradients and the SGD optimisation (see Section A.2.3). A commonly used loss in CNNs is the cross-entropy. Please refer to Section A.2.3 for the forward and backward computation of this loss function.

A.4.4 Regularisation

Several regularisation methods are used in CNNs as described in Section A.2.4. The most common ones include Dropout, DropConnect (randomly setting weights instead of the activations to zero), stochastic pooling, data augmentation, weight decay, early stopping, and ℓ_1 and ℓ_2 regularisation. Also, BN [2] introduced in A.3.4 is used in many recent network architectures.

A.4.5 CNN architectures

Many CNN architectures, learning algorithms, regularisation methods, and applications have been developed and this area is in constant expansion. Some of the most popular CNN architectures used in the literature are described in the following list.

LeNet5

LeNet5 [5] is the first popular CNN architecture trained with backpropagation to recognise digits. The LeNet5 architecture contains two convolution layers, two pooling layers and two fully-connected layers as depicted in Figure A.21. Note that the depth of a CNN refers to the number of layers with trainable parameters (convolution and fully-connected). Thus, LeNet5 is 4 layers deep. The input sizes, number of feature maps and other details are shown in the figure.

AlexNet

AlexNet [15] is a CNN architecture which includes five convolution layers and three fully-connected layers. The main difference from LeNet5 is the depth, the input size

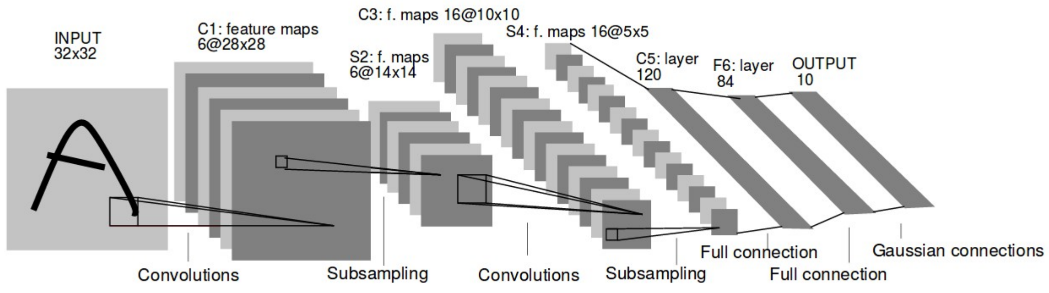


Figure A.21: The LeNet5 architecture. Image replicated from [5].

and the number of neurons, which significantly increased due to the development of computational power and expansion of training data. Dropout and LRN are also incorporated to the network's architecture. The architecture of this network is shown in Figure A.22 and summarised in Table A.1. Note that the channels are grouped into two separate groups which are processed by two different GPUs.

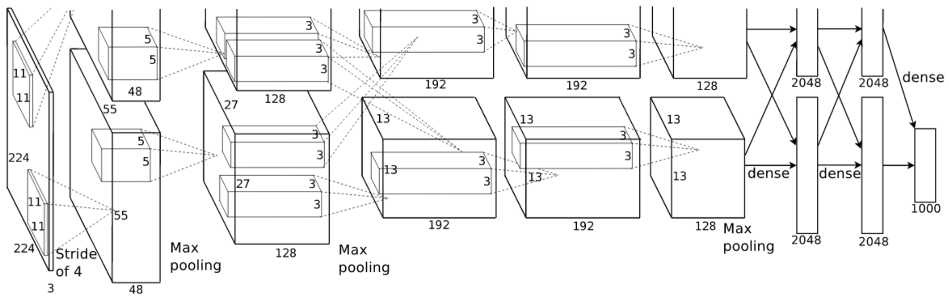


Figure A.22: The AlexNet architecture. Image replicated from [15].

VGG

The VGG-M architecture [18] slightly varies from AlexNet in the kernel size and in the number of feature maps. It contains eight layers like AlexNet (five convolution and three fully-connected layers). The VGG-16 and VGG-19 architectures [19] introduce a more significant change with an important increase of the depth of the network (as indicated by the numbers). Replacing convolution layers of large filter sizes by multiple convolution layers of smaller (3×3) kernel size showed improvement of the CNNs. This approach increases the depth of the network and reduces the number of parameters while keeping the same receptive fields of the neurons. Three convolution layers with 3×3 filters, for instance, have the same receptive field as a single convolution layer with 7×7 filters and less trainable parameters ($3 \times 3 \times 3 + 3 = 28$ parameters versus $7 \times 7 + 1 = 50$ for a single input and output layer). The receptive field of the three 3×3 convolution layers is

Table A.1: AlexNet layers. The convolution and fully-connected layers are all activated by ReLU except for FC3.

Layer type	output size	kernel, pad, stride
Input	$c \times 227 \times 227$	-
Conv (C1)	$96 \times 55 \times 55$	11, 0, 4
Pool (P1)	$96 \times 27 \times 27$	3, 0, 2
LRN	$96 \times 27 \times 27$	-
Conv (C2)	$256 \times 27 \times 27$	5, 2, 1
Pool (P2)	$256 \times 13 \times 13$	3, 0, 2
LRN	$256 \times 13 \times 13$	-
Conv (C3)	$384 \times 13 \times 13$	3, 1, 1
Conv (C4)	$384 \times 13 \times 13$	3, 1, 1
Conv (C5)	$256 \times 27 \times 27$	3, 1, 1
Pool (P5)	$256 \times 6 \times 6$	3, 0, 2
Fully-con. (FC1)	4,096	-
Dropout	4,096	-
Fully-con. (FC2)	4,096	-
Dropout	4,096	-
Fully-con. (FC3)	N	-
Softmax	N	-

illustrated in Figure A.16. The VGG-16 and VGG-19 architectures are successions of multiple 3×3 convolution layers followed by a pooling layer. The large number of feature maps used in these networks (i.e. width of the network) results in a large number of trainable parameters.

Region-based CNN

A Region-based CNN (R-CNN) was developed in [213] for object detection and semantic segmentation. It is based on a region proposal followed by a classification of each region with a large CNN. Therefore, it does not introduce a new network architecture but a method to use an existing trained network to detect objects. The R-CNN framework is described in Figure A.23.

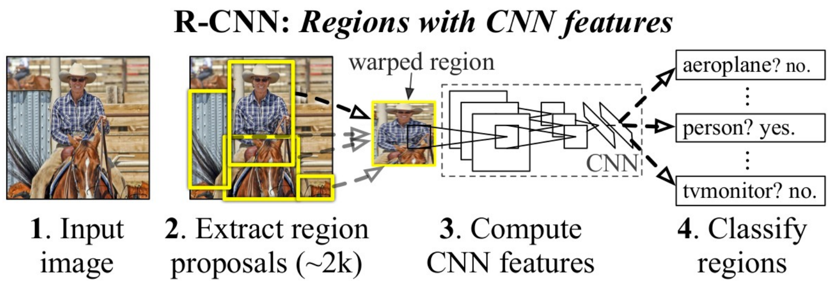


Figure A.23: The R-CNN object detection framework. Image replicated from [213].

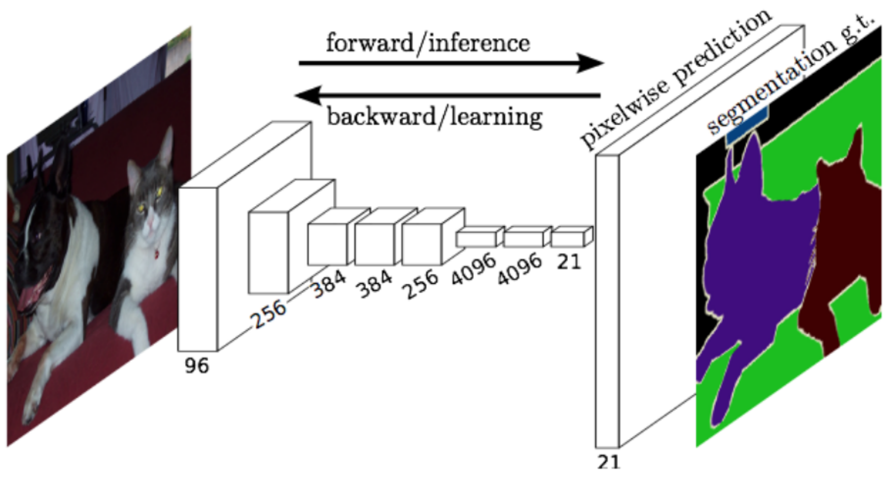


Figure A.24: A basic FCN architecture with pixelwise prediction. The upsampling, deconvolution, and skip layers are not specified and only the prediction image is represented. Image replicated from [213].

Fully convolutional network

A Fully Convolutional Network (FCN) was developed in [6] by adapting classic CNN architectures for semantic segmentation. A FCN is trained end-to-end similarly to classification CNNs except that training images with pixel labels are fed instead of image class labels. An error must be calculated for every pixel location which requires a pixelwise representation of the network's output, unlike a classification output of classic CNNs. To that end, fully-connected layers are first replaced by 1×1 convolution layers to allow arbitrary input sizes. The output feature maps of these 1×1 convolution layers are smaller than the number of input pixels due to convolution and pooling. They are therefore upsampled by deconvolution [205] and interpolation to get back to a pixelwise representation as shown in Figure A.24. Following the idea of hypercolumn in [214], information obtained at multiple depths in the network is combined using skip layers. The local information ("where") in early layers is combined with the global information ("what") in deep layers. This approach allows the network to extract deep features of high complexity while maintaining a locality information crucial for segmentation across boundaries.

Network in network

Based on the idea of stacking simple convolution layers in LeNet5, AlexNet, VGG etc., a new type of architecture emerged in [215] in which the convolution layers are replaced by blocks of operations in "micro" neural networks. In the Network in Network approach (NIN) [215], blocks of 1×1 convolutions (similar to small MLPs) replace the convolution layers to increase the depth and enhance the abstraction ability of the network. This method maintains the local connectivity and weight

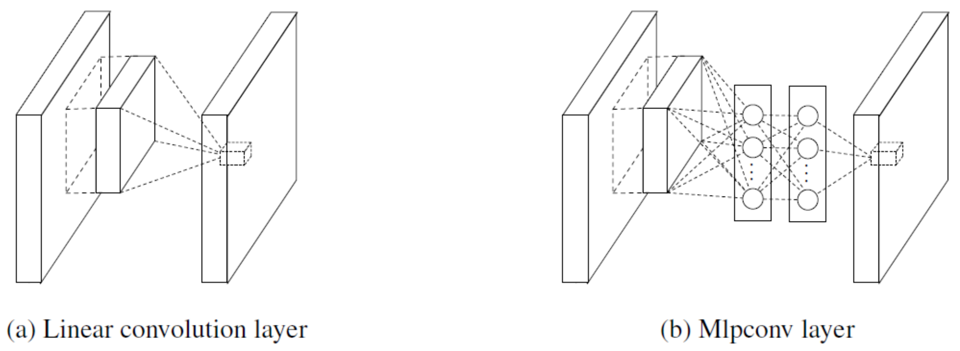


Figure A.25: A comparison of (a) a convolution layer and (b) a Network in Network block (Mlpconv). Image replicated from [215].

sharing of convolution layers. A comparison of a classic convolution layer and this micro network called Mlpconv is illustrated in Figure A.25. Moreover, this block approach and the following ones which will be described replace the max pooling of the last feature map (typically of size 6×6) by an average pooling. In NIN, no fully-connected layer is used to avoid overfitting, that typical consecutive fully-connected layers following a max pooling are prone to in AlexNet and VGG networks. Thus, each feature map in the last Mlpconv layer is generated for a particular class of the training data. Most block CNN approaches based on this work, as described in the following sections, maintain the average pooling method yet use a single fully-connected layer for convenience in adapting the networks to different label sets.

GoogleNet and Inception modules

A new concept named "Inception module" was introduced in [164] (Inception-v1) based on the block approach of the NIN. An Inception module processes the data at multiple scales as illustrated in Figure A.26. It applies a set of convolutions with different kernel sizes and a pooling operation in parallel and combines the outputs. A trick of dimension reduction (with 1×1 convolutions inspired by NIN) before the convolution with large kernels enables a significant reduction of computation complexity. This is commonly referred to as *bottleneck* as information is "squeezed" into a lower dimension. GoogleNet is a 22 layers network including three convolution layers, nine Inception modules (two layers each), and one fully-connected layer. The number of feature maps is up to 1,024 in the last Inception module. The full architecture of GoogleNet is illustrated in Figure A.27. To efficiently train GoogleNet, the error is calculated at multiple intermediate layers.

Several variants of the Inception module were proposed following this initial work. Batch-normalisation was added to the Inception module in [2] (Inception-v2). In Inception-v3 [216], the convolutions are factorised into smaller convolutions.

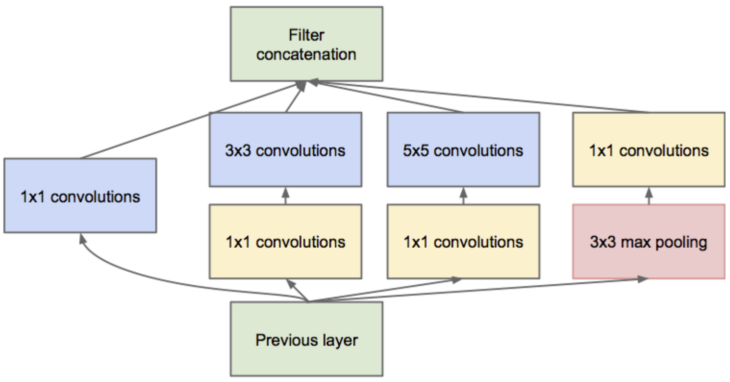


Figure A.26: An inception module used in GoogleNet. Image replicated from [164].

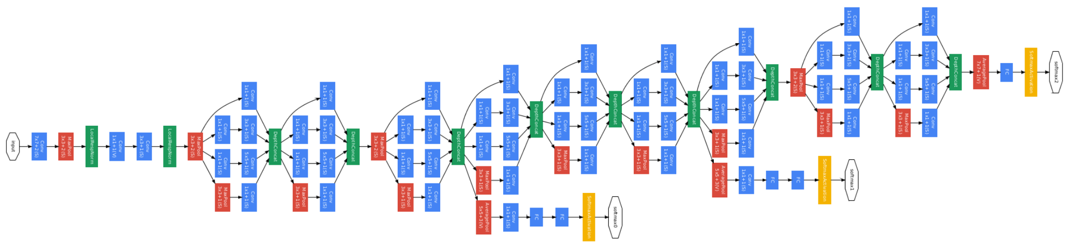


Figure A.27: The GoogleNet architecture. Convolution layers are depicted in blue, pooling layers in red, softmax in yellow, concatenation and normalisation in green and finally, input and labels in white. Image replicated from [164].

5×5 convolutions can be replaced by two 3×3 convolutions as suggested in the VGG networks and to go further, 3×3 convolutions are replaced by asymmetric 3×1 followed by 1×3 convolutions for a more efficient implementation.

Deep residual networks

Residual connections were introduced in [210] to train very deep networks. Deep residual networks are significantly deeper than the previously discussed CNNs with up to more than 1,000 layers. It was shown that adding more layers to classic CNNs results in a saturation and rapidly a degradation of the accuracy. This is not due to overfitting but to the difficulty to optimise deeper architecture.

Rather than approximating a function $H(\mathbf{x})$, a residual approach approximates a residual function $F(\mathbf{x}) := H(\mathbf{x}) - \mathbf{x}$. The resulting original function $F(\mathbf{x}) + \mathbf{x}$ is implemented by adding a “shortcut” connection as shown in Figure A.28. Note that this shortcut connection is frequently compared to an LSTM unit without gates. A deep ResNet is then created by stacking many residual learning blocks with mainly 3×3 convolution filters. This residual approach allows the data to flow from one block directly to a deeper block through the shortcut connections. Similarly,

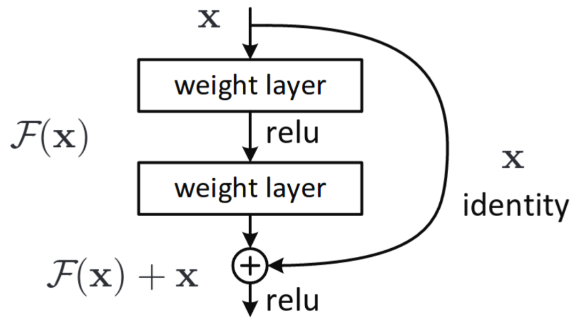


Figure A.28: A residual learning block. Image replicated from [210].

gradients propagate directly from any block to shallower ones in the backward pass. Therefore, very deep residual networks can be efficiently optimised from scratch without the error calculation at multiple levels adopted in GoogleNet.

Intuitively, the residual block is motivated by the fact that it performs an identity mapping if the weights of the convolution layers are zeros ($F(\mathbf{x}) + \mathbf{x} = \mathbf{x}$). Adding an identity mapping cannot degrade the performance of the network, as opposed to adding classic layers. A residual block will therefore learn a residual if it results in a gain of accuracy, or drive the weights towards zero to perform the identity mapping.

Originally directly attributed to the increased depth [210], the reason for the performance of deep ResNets is currently subject to broad interest [217]. This discussion is beyond the scope of the thesis.

Other block approaches

More block-based approaches were developed in the literature based on the NIN, Inception and ResNet architectures including Inception-v4 and Inception-ResNet [218], SqueezeNet [219] and X-ception [220].

Convolutional generative adversarial networks

This section presents an application of the GAN introduced in Section A.3.6 with convolutional networks. In a convolutional GAN [9], the generator is a deconvolutional network [205] which generates images. The discriminator is a convolutional network which performs a binary classification to predict whether an input image is generated ("fake") or a sample from the training dataset ("real").

In the Deep Convolutional Generative Adversarial Network (DCGAN) [221], the generator and discriminator are equipped with recent developments in supervised CNNs including BN (see Section A.3.4) and strided convolutions [212]. DCGANs can generate very realistic looking images and learn unsupervised representations which have been successfully used as a feature extractor for image classification.

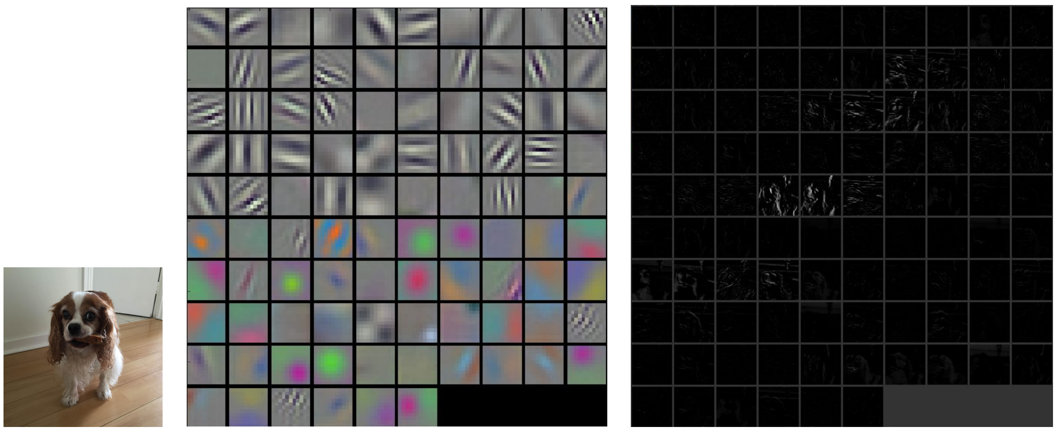


Figure A.29: A visualisation of filters and responses of the neurons in the first convolution layer of AlexNet trained on ImageNet. (a) Input image, (b) filters, (c) responses. Figures obtained with the DeepVis toolbox [170].

A.4.6 Applications

Convolutional networks have been highly successful in most machine learning problems, in particular in computer vision for the analysis and synthesis of data from various modalities. CNNs can be applied to data of multiple dimensions, e.g. 1D audio signals, 2D images, 3D volumetric data or videos and higher-dimensional medical data. Single or multiple input channels can be used, multiple modalities can be integrated and recent architectures allow various input sizes. This flexibility, together with its performance and the computational power and scalability which arose from recent GPUs, have made CNNs one of the most popular approaches in machine learning in the last decade. An exhaustive list of the extensively growing number of applications is outside the scope of this thesis. Some of the major applications in computer vision include image recognition, object detection and tracking, pose estimation, text detection and recognition, visual saliency, action recognition, and scene labelling. Other machine learning applications include speech processing, natural language processing, and text classification. A survey of CNN applications can be found in [222].

A.4.7 Visualisation

Several methods have been developed to visualise and analyse what and how CNNs learn. The simplest method is to visualise the feature maps and weights at different layers in the network. Figure A.29 shows the weights of the first layer of AlexNet trained on ImageNet as well as the feature maps of the first layer in response to an input image.

Most filters have learned to detect edge-like features similar to a Gabor filter bank approach and to the primary visual cortex feature detection. Other filters have learned colour blobs and edges. Note that the visualisation of the weights is mainly interpretable for the first layer as the feature maps of deeper layers are a result of a combination of multiple filters in previous layers.

For this reason, more advanced methods were developed to visualise what the neurons in intermediate layers actually perceive from the input images. Recall that the receptive field grows as one navigates towards the output of the network and neurons learn to respond (i.e. fire) to larger and more complex patterns. Three methods will be introduced here to visualise what the neurons detect in the input image.

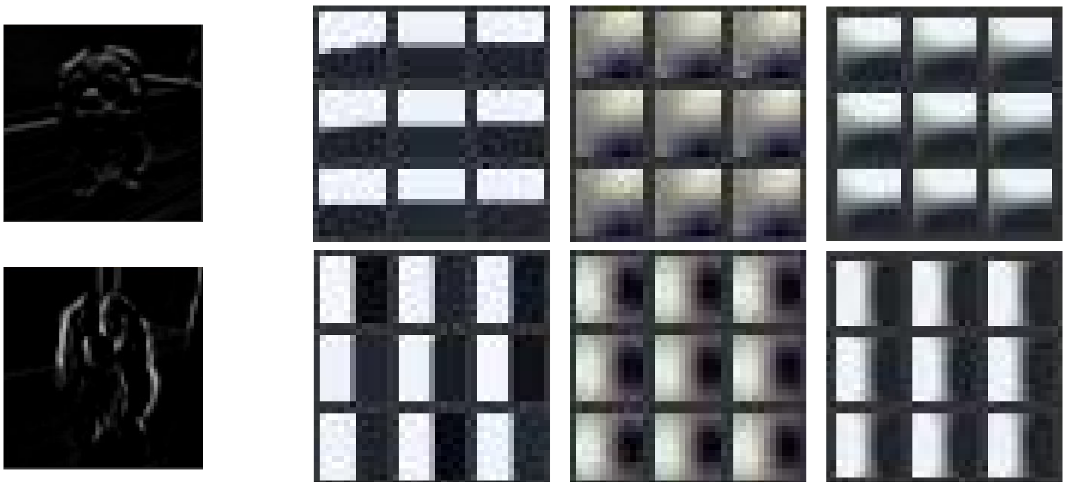
The first method is to simply find which input image maximally activates a certain neuron. In other words, all images from a dataset (generally unknown to the network) are used one by one as input to the CNN and the one for which the activation of the neuron of interest is maximal is selected. This is a very basic method which is most relevant for neurons in high layers. Figure A.30 (second column) shows image patches which maximally activate two neurons in the first, second and fifth layers of AlexNet trained on ImageNet.

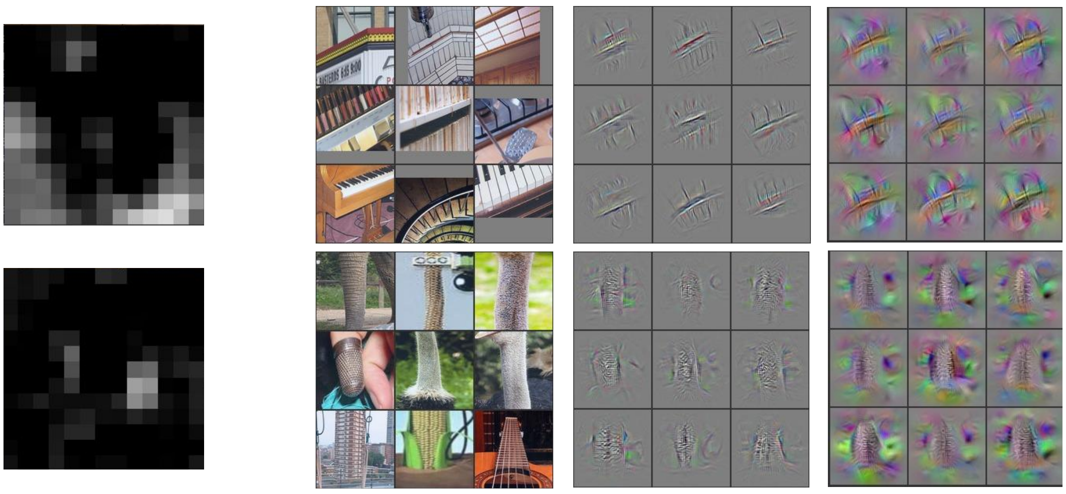
Another approach is based on the deconvolution [211]. The idea is to reconstruct an input image by reversing the operations of the CNN. To visualise what a neuron at a certain layer has learned to detect, the activation that fires most with unknown input images is found and the operations are reversed to project the activation back into the pixel space. In particular, deconvolution, reversed max-pooling, and reversed ReLU operations are used. Note that the deconvolution pass uses the same convolutional kernel and pooling "switches" as the forward pass. Figure A.30 (third column) shows the deconvolution visualisations of the same neurons as the first visualisation method.

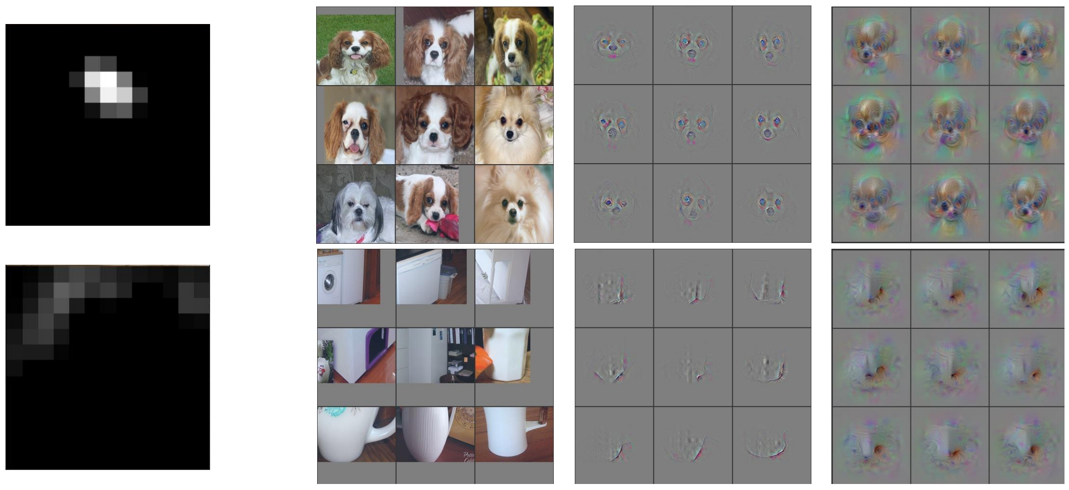
The last visualisation method presented here involves the generation of an image (a receptive field) which maximally activates a neuron [169]. To that end, the input values are updated in a gradient ascent optimisation to maximise the activation of a neuron as a function of the inputs of all layers up to the input image. Figure A.30 (last column) illustrates the generated receptive fields for the same neurons as the previous visualisation methods.

Figure A.29 and A.30 illustrate that deep neurons respond to complex patterns and enable to analyse which patterns are sought by the neurons. For instance, the neurons in the first convolution layer mainly detect edges and colours. Neurons in the third layer respond to more complex patterns such as a cylinder-like shape (dog's leg) or a texture with straight edges like the parquet floor. Finally, the neurons in the last convolution layer respond to complex patterns such as the dog's face, and learn some type of invariances and abstraction necessary to image recognition. Various interesting analyses and conclusions can be drawn from such visualisation methods.

(**a**) input image


(**b**) Conv1


(**c**) Conv3



(d) Conv5

Figure A.30: A visualisation of several features learned by neurons in AlexNet trained on ImageNet and their response to an input image. The first column depicts the response of a particular neuron. The second column shows the image patches which maximally activate this neuron. The third and fourth columns show the deconvolution images and the images obtained by activation maximisation respectively. (a) input image, (b) two neurons in Conv1, (c) two neurons in Conv2, (d) two neurons in Conv5. Figures obtained with the DeepVis toolbox [170]. Note that the deeper receptive fields are larger than shallow ones but are resized for display.