

Linguistic Interpretation of Visual Contents via Deep Learning

Von der Fakultät für MINT - Mathematik, Informatik, Physik,

Elektro- und Informationstechnik

der Brandenburgischen Technischen Universität Cottbus–Senftenberg

zur Erlangung des akademischen Grades eines

Dr.-Ing.

genehmigte Dissertation

vorgelegt von

M.Sc.

Ali Sharifi Boroujerdi

geboren am 1.4.1981 in Teheran, Iran

Gutachter: Prof. Dr. rer. nat. habil. Michael Breuß

Gutachter: Prof. Dr.-Ing. Ingo Schmitt

Gutachter: Prof. Dr. Thomas Brox

Tag der mündlichen Prüfung: 15.6.2018

Abstract

The main part of the research outlined in this thesis is to develop deep learning models for the linguistic interpretation of the visual contents. This part is split into two research problems: interactive region segmentation and captioning, and selective texture labeling. In the first attempt, we proposed a novel hybrid deep learning architecture whereby the user is able to specify an arbitrary region of the image that should be highlighted and described. The proposed model alternates the bounding box indications of the standard object localization process with the output of a deep interactive segmentation module to achieve a better understanding of the dense image captioning and improve the object localization accuracy. The idea of the next part is to establish a bidirectional correlation between deep texture representation and its linguistic description via a hybrid CNN-RNN model that enables end-to-end learning of the selective texture labeling. This novel architecture provides new opportunities to describe, search, and also retrieve texture images from their linguistic descriptions. To be able to train such a model, we generated a multi-label texture dataset that covers color, material, and pattern labeling simultaneously. Our contribution to the automatic generation of texture descriptions provides an excellent opportunity to enrich the existing vocabulary of the image captioning. Such a conceptual extension can be used for fine-grained captioning applicable in geology, meteorology and other natural sciences where fine-grained image structures are of importance to understand complicated patterns. Apart from deep learning technologies, in the final section of the thesis, we proposed a novel approach to define mathematical morphology on color images. To this end, we converted common RGB-values of the color images into a new biconal color space and then combined two approaches of mathematical morphology to give meaning to the maximum and the minimum of the matrix field data and formulate our novel strategy.

Deutsche Zusammenfassung

Der wesentliche Teil dieser Arbeit besteht darin, Deep Learning (DL) Modelle für die sprachliche Interpretation von visuellen Inhalten in Bildern zu entwickeln. Hierzu werden zwei grundlegende Aufgabenstellungen verfolgt: Zum einen die interaktive Segmentation und das sogenannte Captioning, bei dem Objekte sprachlich beschrieben werden, zum anderen das selektive Labeling von Texturen. Zur Lösung der ersten dieser Aufgabenstellungen wird eine neue, hybride DL-Architektur vorgestellt, bei der der Benutzer eine beliebige Region eines Eingabebildes anklicken kann, die automatisch segmentiert und sprachlich beschrieben werden soll. Das neue Modell alterniert den Indikator der Bounding Box eines üblichen Verfahrens zur Objektlokalisierung mit der Ausgabe eines interaktiven, DL-basierten Segmentationsmodules. Auf diese Weise wird ein besseres Resultat für das Captioning erzielt, und auch die Objektsegmentierung wird verbessert. In Bezug auf den zweiten Teil der neuen Entwicklungen wird eine wechselseitige Korrelation zwischen einer DL-basierten Texturrepräsentationen einerseits und deren sprachlicher Beschreibung andererseits mittels einer neuen, speziellen Konstruktion neuraler Netze hergestellt. Dieses neuartige, hybride CNN-RNN-Modell ermöglicht ein sogenanntes end-to-end Learning von Texturbeschreibungen. Wie aufgezeigt wird, entstehen durch die spezielle Architektur dieses neuralen Netzes neue Möglichkeiten zur Charakterisierung, gezielten Suche und zum Retrieval von texturierten Bildern mittels sprachlicher Beschreibungen als Eingabe. Um ein solches Modell zu trainieren wurde ein Texturdatensatz erstellt, das gleichzeitig Informationen über Farbe, Material und eine Musterbeschreibung enthält. Der Beitrag dieser Arbeit zur automatischen Generierung von Texturbeschreibungen in Eingabebildern ermöglicht durch die Flexibilität des Ansatzes neue Möglichkeiten der Texturbeschreibung für viele potenzielle Anwendungen. So erscheint es beispielsweise gut möglich, detaillierte Beschreibungen von Texturen für Bilder aus Bereichen wie Geologie, Bodenkunde und anderen Naturwissenschaften, in denen eine feine Bildtextur eine erhebliche Rolle spielen, mittels dieses Ansatzes zu ermöglichen. In einem weiteren Teil der Arbeit wird ein neuartiger Ansatz für die Realisierung von morphologischen Prozessen auf Farbbildern vorgestellt. Hierzu wird der RGB-Farbraum in einen Farbdoppelkegel transformiert, in dem man mathematische Operationen wie die Berechnung eines Maximums oder Minimums gegebener Farben für die Berechnung solcher Prozesse theoretisch untermauert umsetzen kann.

Acknowledgments

There are many people I must thank for contributing to the four wonderful years of my experience as a PhD student. First and foremost, I would like to thank my dearest *Maryam* who has contributed to my day-to-day life. Without her encouragement and persistent help this achievement would not have been possible. I would also like to give special thanks to *Prof. Dr. Michael Breuß* for giving me the opportunity to pursue this research program and for his support and guidance throughout the past four years. I also appreciate his kind family for being supportive since the days we began our new life in Germany.

Furthermore, I wish to express my sincere gratitude to Prof. Dr. Ingo Schmitt and Prof. Dr. Thomas Brox for reviewing this thesis. I am also grateful for my committee members, Prof. Dr. Ekkehard Köhler and Dr. Laurent Hoeltgen, for their services and helps. I am fortunate to have had such a great committee.

I would also like to extend thanks to my friends who I had the pleasure of interacting with and learning from during my PhD, Prof. Rembert Reemtsen and Prof. George Bader for their friendly conversations, Dr. Friedemann Kemm for his fascinating knowledge, my former colleague Dr. Andreas Kleefeld and his nice family, my technical colleague Mrs. Büttner for all her supports and kindness, our secretary Mrs. Kallweit, Dr. Yvain Quéau, Mehran Amini, Martin Bähr, Georg Radow, Robert Dachsel, Hakam Kondakji, Amin Naif, Saman Amanpour, Mehrdad Mirzapour, Sahel Ramezani, Mohammad Mozooni, Rojin Irannejad, Sara Riazi, Isabelle, David, Mohammad, Babak, Bahman, Neda, Amir Hosein, Amir Mohsen and many more. I have also learned a lot from people I have never met, specially Andrej Karpathy and Justin Johnson at Stanford University, so I thank them all.

Finally, I owe my deepest gratitude to my parents *Parviz, Soraya, Mostafa* and *Tooran* for their endless love, support and engagement.

Journal Publications and Pre-prints

- Maryam Khanian, Ali Sharifi Boroujerdi, Michael Breuß (2018), “Photometric Stereo for Strong Specular Highlights”, Computational Visual Media, Springer.
- Martin Bähr, Michael Breuß, Yvain Quéau, Ali Sharifi Boroujerdi, Jean-Denis Durou (2017), “Fast and Accurate Surface Normal Integration on Non-Rectangular Domains”, Computational Visual Media, Springer.
- Michael Breuß, Laurent Hoeltgen, Ali Sharifi Boroujerdi, Ashkan Mansouri Yarahmadi (2016), “Highly Robust Clustering of GPS Driver Data for Energy Efficient Driving Style Modeling”, arXiv preprint, arXiv:1610.02815.
- Maryam Khanian, Ali Sharifi Boroujerdi, Michael Brauß (2018), “Real-Time 3D Shape for Micro Details”, arXiv preprint, arXiv:1802.06140.

Conference Publications

- Ali Sharifi Boroujerdi, Maryam Khanian, Michael Breuß (2017), “Deep Interactive Region Segmentation and Captioning”, *International Conference on Signal Image Technology and Internet-based Systems (SITIS)*. †
- Ali Sharifi Boroujerdi, Michael Breuß, Bernhard Burgeth, Andreas Kleefeld (2015), “PDE-Based Color Morphology Using Matrix Fields”, *International Conference on Scale Space and Variational Methods in Computer Vision (SSVM)*. †
- Maryam Khanian, Ali Sharifi Boroujerdi, Michael Breuß (2015), “Perspective Photometric Stereo Beyond Lambert”, *International Conference on Quality Control by Artificial Vision (QCAV)*.

†publications that are directly related to this dissertation

Contents

1	Introduction	23
1.1	Motivation of the Thesis	23
1.2	Natural Vision	24
1.3	Machine Vision	28
1.4	Outline of the Dissertation	31
2	Deep Learning Background	33
2.1	Learning Approaches	35
2.1.1	Supervised Learning	35
2.1.2	Unsupervised Learning	36
2.1.3	Reinforcement Learning	37
2.2	Artificial Neural Networks	38
2.2.1	Feed-Forward Networks	39
2.2.2	Output Layers and Loss Functions	40
2.2.3	Training Terminology	41
2.2.4	Depth Influence	42
2.3	Backpropagation	42
2.4	Optimization	46
2.4.1	Gradient Descend	46
2.4.2	Some Optimization Catalysts	48
2.5	Overfitting and Regularization	49
2.5.1	Dropout	50
2.5.2	Batch Normalization	52

2.6	Convolutional Architectures	52
2.6.1	LeNet	55
2.6.2	AlexNet	56
2.6.3	ZFNet	56
2.6.4	VGGNet	57
2.6.5	GoogLeNet	59
2.6.6	ResNet and Residual Learning	60
2.6.7	DenseNet	62
2.6.8	Fully Convolutional Networks (FCNs)	63
2.7	Representation Learning	64
2.8	Feature Encoding	66
2.9	Transfer Learning and Fine-tuning	67
2.10	Region Based CNNs	68
2.10.1	R-CNN	69
2.10.2	Fast R-CNN	71
2.10.3	Faster R-CNN	71
2.11	Sequence Modeling	73
2.11.1	Autoregressive Models	74
2.11.2	Hidden State Models	74
2.11.3	Recurrent Neural Networks	76
2.11.4	LSTM	78
2.12	Deep Learning Frameworks	80
2.12.1	Caffe	80
2.12.2	Theano	80
2.12.3	Tensor Flow	81
2.12.4	Torch	81
2.12.5	DL4J	83
3	Deep Interactive Region Segmentation and Captioning	85
3.1	Introduction	85

3.2	Deep Region Detection	86
3.2.1	Generic Input Representation	88
3.2.2	Deep Region Proposals	89
3.3	Automatic Image Description	94
3.3.1	Generative Captioning Models	94
3.3.2	Retrieval-based Captioning Approaches	96
3.3.3	Recurrent Language Model	99
3.4	Deep Interactive Segmentation	102
3.4.1	User Interaction Imitation	104
3.4.2	Morphological Cortex Detection (MCD).	107
3.4.3	Fully Convolutional Interactive Segmentation Networks	107
3.4.4	Fusion Approach	109
3.5	Experimental Results	111
3.5.1	Datasets	111
3.5.2	Preprocessing	111
3.5.3	Training Process of the ReCap and Interactive Segmentation Modules	112
3.5.4	Segmentation Accuracy Metrics	113
3.5.5	Test of Localization Accuracy	114
3.5.6	Sensitivity Analysis	117
3.5.7	Quantitative Comparison of the Segmentation Quality	118
3.5.8	Dense Interactive Region Captioning	118
4	Deep Selective Texture Labeling	129
4.1	Introduction	130
4.1.1	Classic Texture Representation	131
4.1.2	Vocabulary Learning	132
4.2	Deep Texture Representation	134
4.3	Our Approach for Selective Texture Labeling	135
4.3.1	Connectivity Plan of the Spectator Modules	137
4.3.2	Bilinear Texture Representation	139

4.3.3	Semantic Label Generation	140
4.4	Experiments	141
4.4.1	Multi-Label Texture Dataset	141
4.4.2	Training and Optimization	146
4.4.3	Qualitative Results	146
4.4.4	Model Performance Evaluation	147
4.5	Conclusion	156
5	PDE-based Color Morphology using Matrix Fields	157
5.1	Introduction	158
5.2	PDE-based Morphology	160
5.3	Numerical Methods for the PDEs of Dilation/Erosion	162
5.4	Color Images and Matrix Fields	165
5.5	Pseudo Supremum/Infimum and Functions of Matrices	166
5.6	Experimental Results	168
6	General Conclusion	175
6.1	Summary of Contributions	175
6.2	Future Directions	176

List of Figures

2-1	The general structure of a biological neuron [3]. The figure is redrawn based on [4]	38
2-2	The general structure of an artificial neuron [3].	39
2-3	Error derivatives calculation by backpropagation [28]	45
2-4	A demonstration of an elongated problem space. The figure is redrawn based on Figure 2.1 in [104]	48
2-5	The general architecture of a conv/pooling building block in a CNN [28] . .	53
2-6	LeNet architecture [115], [3]	56
2-7	AlexNet architecture [107], [28]	57
2-8	Visualization of a low-level convolutional layer as proposed in [209]. The figure is redrawn based on the figure 6 (c) of [209]	58
2-9	A repaint [28] of VGGNet [184] with 16 layers including five conv/pooling blocks and three FC layers	59
2-10	Internal structure of an inception module applied in GoogLeNet [190]. The figure is redrawn based on the figure 2 (a) of [190]	61
2-11	The structure of a residual block as applied in ResNet [81]. The figure is redrawn based on the figure 2 of [81]	62
2-12	DenseNet [88] structure. The figure is redrawn based on the figure 2 of [88]	63
2-13	A repaint [28] of a Fully Convolutional Network (FCN) architecture [131] including a learnable upsampling mechanism	64
2-14	A repaint [28] of R-CNN mechanism [70] for object recognition	70
2-15	A repaint [28] of Fast R-CNN framework [69]	72

2-16	A repaint [28] of the anchor generation on CNN feature maps in Faster R-CNN approach [163]	73
2-17	Unrolling of RNN [28]	77
2-18	A repaint [28] of LSTM gating mechanism [84]	79
3-1	(a) Input image [3] including positive and negative user clicks, (b) output of the dense captioning process [95], (c) probability map of our deep interactive segmentation framework considering user priorities, and (d) the final output of our hybrid model including highlighted user intended region (UIR) and its proper description.	87
3-2	The recognizer part of the proposed Recognition and Captioning (ReCap) module [28].	103
3-3	Input image including positive and negative clicks (up), and obtained positive (middle) and negative (bottom) Voronoi diagrams [28].	105
3-4	Our proposed Morphological Cortex Detection (MCD) technique [28].	108
3-5	Architecture of the proposed deep interactive region segmentation and captioning model [28].	110
3-6	Operating progress of our proposed hybrid model, from left to right: input images obtained from PASCAL VOC 2012 dataset [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.	115
3-7	Localization accuracy comparison between our model (LFCN8s) and the internal region proposal network (RPN) of the ReCap module (up), and mean IoU accuracy of our proposed model (LFCN8s) on several segmentation benchmarks against different number of clicks (bottom) [28].	116
3-8	Model sensitivity analysis against number of clicks on a sample input image obtained from PASCAL VOC 2012 dataset [62] (up), and its associated IoU rates (bottom). Here, the IoU measure is computed between the segmentation result and the ground truth mask of the object.	117

3-9	Segmentation quality comparison between our three proposed interactive segmentation modules and other interactive segmentation techniques. Our method clearly provides more abstract region and object understanding. Input images are obtained from PASCAL VOC 2012 dataset [62]	119
3-10	Output probability map (left) and the final result (right) of our approach for different regions of an image in response of different user interactions [28].	120
3-11	Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.	124
3-12	Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.	125
3-13	Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.	126
3-14	Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.	127

3-15	Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.	128
4-1	Each texture pattern can be located somewhere between regularity (left) and randomness (right). Images are obtained from DTD [43] and ALOT [37] datasets.	130
4-2	Architecture of the proposed deep selective texture labeling model [28]. . .	138
4-3	Some samples of ALOT dataset [37] (back) including material labels such as <i>sunflower seeds</i> , <i>sheep wool</i> and <i>spaghetti</i> , and DTD dataset [43] (front) providing pattern labels such as <i>banded</i> , <i>crystalline</i> , and <i>marbled</i> that are used with color information to shape our <i>multi-label texture dataset</i>	142
4-4	Extraction of the dominant color's name via grid sampling and average pooling based on CSSW3C standard [28].	143
4-5	Each training sample includes a positive and a negative texture that are combined based on one of different arrangements determined by randomly generated binary masks (black is positive) [28].	145
4-6	Some qualitative results of the proposed model. From left to right: test samples obtained from [113], [111], or [78], corresponding binary masks (black is positive), positive textures in test samples, ground truth descriptions (in black), and predicted descriptions by the proposed model (in blue).	148
4-7	False samples of the proposed model. From left to right: test samples obtained from [113], [111], or [78], corresponding binary masks (black is positive), positive textures in test samples, ground truth descriptions (in black), and predicted descriptions by the proposed model (in red).	149
5-1	Color bi-cone, figure adapted from [36]	165

5-2	(Center) Input image yin-yang [5] defined using <i>rgb</i> values for black/white. (Left) Ten times dilation with color-valued FCT. (Right) Ten times erosion with color-valued FCT.	168
5-3	(Top row) Original <i>Lenna</i> image [2], and results of dilation and erosion computed with the RT scheme. (Bottom row) Morphological Laplacian and results of five and ten iterations of shock filtering with the RT scheme.	169
5-4	Input image for comparison of numerical methods [29]	170
5-5	Results of eight time steps dilation with $\tau = 1/2$ for indicated PDE-based schemes and in accordance four times erosion of BK method [29].	171
5-6	Original image of size 256×256 (left) and result after five iterations of dilation using FCT with $\tau = 1/2$ and a disk-shaped structuring element (right). Black frames indicate new colors that appear by use of the supremum rule and the yellow ones mark color interaction without new colors [29].	172
5-7	Colors in the example [29]	172

List of Tables

3.1	Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on Berkeley (BSDS500) dataset [136].	121
3.2	Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on Weizmann dataset [13].	121
3.3	Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on Alpha Matting dataset [165].	122
3.4	Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on MOS dataset [183].	122
3.5	Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on VOC validation dataset [62].	123
4.1	Label conversion protocol between our multi-label dataset and UIUC [113] dataset.	153
4.2	Label conversion protocol between our multi-label dataset and Kylberg [111] dataset.	154
4.3	Label conversion protocol between our multi-label dataset and KTH-tips-2b [78] dataset.	155

4.4 Diverse captioning scores of our proposed model on three different external datasets for full-image non-selective texture labeling experiments. 155

4.5 Diverse captioning scores of our proposed model on three different external datasets for selective texture labeling experiments on dual texture combinations. 155

Chapter 1

Introduction

1.1 Motivation of the Thesis

Our current age has been given many names, as early as 70's Bell [24] talked about the post-industrial society. Afterwards, terms such as information society, digital age, and network society were utilized to explain our rapidly changing global village. The basic idea is that previously separated technologies converge together on what we call information technology that is used to extract, store, manipulate and distribute information from structured and unstructured sources of data. In this situation, the world storage capacity has grown at about 25% per year over the recent decades while our telecommunication capacity witnessed an even more annual growth of 30%. Most impressively, the world's computation capacity is amplified twice as fast and general purpose computers grew at about 60% percent per year [82]. These changes are extremely fast in comparison with the changing rates that we are used to such as the average annual growth rate of the global economy.

Increasing presentation of a wide variety of contents in digital form results in easier and cheaper duplication and distribution of information. Every day hundreds of millions of people take photos, make videos and send texts. Across the globe, businesses collect data on consumer preferences, purchases, and trends. Governments regularly collect all sorts of data from census data to incident reports in police departments. According to latest Cisco visual networking index [160], by 2021 annual global IP traffic will reach 3.3 ZB, global IP traffic will increase nearly threefold, smartphone traffic will exceed PC traffic, traffic from

wireless and mobile devices will account for more than 63% of total IP traffic, the number of devices connected to IP networks will be three times as high as the global population and broadband speeds will nearly double.

All these statistics are telling a common story: the amount of the available data grows fantastically. The majority of this information consists of human-produced unstructured data such as images and videos. Visual contents drive an engagement that has an enormous impact on the audience and influences human emotion. So, it is of crucial importance to develop techniques for multimedia information indexing and retrieval to establish a lingual interaction between people and the visual contents.

1.2 Natural Vision

Light is the most valuable source of energy and knowledge presented in the nature. Plants and other self-feeding creatures produce their own food by converting carbon dioxide, water, and light energy into carbohydrates during a process known as photosynthesis. Moreover, most of the animals understand their surrounding environment using light in the visible spectrum reflected by the objects. For this purpose, their visual system including eyes, optic nerves, and brain translates varying directions and energies of the photons into different kinds of information such as shapes, colors and brightness.

Eyes are sensory organs that detect light and convert it into electrochemical impulses in neurons. In microorganisms, eyes do nothing but detect whether the surroundings are light or dark, while in higher organisms, eye is a complex optical system that collects light from the surrounding environment, regulates its intensity through a diaphragm, focuses it through an adjustable assembly of lenses to form an image, converts this image into a set of electrical signals, and transmits these signals to the brain through complex neural pathways that connect the eye via the optic nerve to the visual cortex and other areas of the brain.

Generally speaking, eyes can be categorized into two main sub-branches: simple and compound eyes. Simple eyes normally consist of a seamless photoreceptive surface. Although the name suggests simplicity, simple eyes are not simple in photosensitivity and accuracy but only in the structure. Simple eyes are found in many species among vertebrates and

invertebrates. There are few types of simple eyes known as pit eyes, spherical lens eyes, multiple lenses, refractive cornea, and reflector eyes. Pit eyes are the most primitive of all types of eyes, and there is a small depression with a collection of photoreception cells. It is important to notice that pit vipers have pit eyes to sense the infrared radiation of their prey animals. The spherical lens eyes have a lens in the structure, but the focal point is usually behind the retina, causing a blurred image to detect the intensity of light. Multiple lenses simple eyes are an interesting type with more than one lens in a shape of a telescope, which enables the observer to enlarge the picture and get a sharp and focused image. Certain predators such as spiders and eagles are good examples of this type of lens arrangement. The eyes with a refractive cornea have an outer layer of a light penetrating substance, and the lens is not usually spherical, but its shape could be changed according to the focal lengths. Reflector eyes are a wonderful phenomenon that provides a common communication platform for other organisms, as well. The image formed in one's eye is reflected onto another place so that the other organisms could see it. All these types of simple eyes function in taking the information with regard to light to sustain the body. Despite all these being simple eyes, all the higher vertebrates including humans have simple eyes. Compound eyes are formed by repeating the same basic units of photoreceptors called ommatidia. An ommatidium has a lens and photoreceptive cells mainly, and the pigment cells separate each ommatidium apart from the neighbors. However, compound eyes are capable of detecting motions as well as the polarization of sunlight, in addition to receiving light. The insects, especially honeybees have the ability to understand the time of the day using the polarization of the sunlight from their compound eyes. There are few types of compound eyes known as apposition, superposition, parabolic suspension, and some few more kinds. Within these species, visual information is formed through ommatidia taken into the brain, and the whole image is combined there in order to understand the object in the apposition eyes. The superposition eyes form the image by reflecting or refracting the light received via mirrors or lenses, and then the image data are transferred into the brain, to understand the object. The parabolic suspension eyes use the principles of both apposition and superposition eyes. Most of the annelids, arthropods, and molluscs have compound eyes, while they can see colors, as well.

During the long process of the evolution, various visual systems are adjusted to different requirements, environments, and behaviors. For instance, dragonflies have been around for more than 300 million years. In this time, they fine-tuned their eyes to see the world in the slow-motion. Their vision is so quick that it can track a flying object in less than five-hundredths of a second. It is partly due to the speed that they process information. Dragonfly's brain is able to process 200 individual frames per second three times faster than a human brain. Moreover, they have about 30,000 ommatidia per eye that can see the ultraviolet and polarized light, as well as three smaller eyes capable of detecting movement, allowing them to react within a fraction of a second.

As another consequence of the long-term evolution, eyes may be mounted on stalks to provide a better all-around vision that also allows the animal to track prey and predators without moving the head.

Despite their fantastic functionality, all types of natural visions are limited to a small range of electromagnetic spectrum that leads to different levels of color perception. Small changes to the genes coding for the most sensitive pigment known as rhodopsin is able to change the color perception drastically.

While eyes' operation is truly complicated, they are just the beginning of an even more complex series of processing stages conducted in the brain. At the beginning of this multi-step process, the retina converts the 3D objects view into 2D images. The retina is actually a part of the brain that is isolated to serve as a transducer for the conversion of light into neuronal signals. Afterwards, the brain rebuilds a 3D model of the world based on the information in the 2D image. Almost forty percent of the human brain is involved with visual data processing [19]. The most dedicated part of the brain that is located in the back of the two hemispheres is the primary visual cortex that is usually referred as V1. This neural region is the biggest area between around 30 identified active visual brain sections. The connections between retina cells and cortical neurons are topographic in that nearby areas on the retina surface correspond to nearby visual cortex regions. The correspondence between the retina and cortical regions can be formulated as a log-polar transformation, in which standard axes in the retina are mapped into polar axes in the cortex. Neurons in the visual cortex fire action when visual stimuli appear within their receptive field. The

logarithmic component of the transformation accounts for the magnification of central representations in the cortex. Thus, this transformation from the retina to cortex preserves the qualitative spatial relations but distorts quantitative ones. Hubel and Wiesel [89] categorized visual cortex neurons in their pioneering investigations into simple, complex and hypercomplex cells. Images need to have a particular orientation in order to excite a simple cell, while a correctly oriented bar of light might need to move in a particular direction to stimulate a complex cell and in hyper-complex cases, the bar might also need to be of a particular length.

In addition, there are several neural pathways that are used to process different visual properties such as color, shape, motion, and depth. Motion processing includes different kinds of information such as the derivation of the speed and direction of a moving target, the motion boundaries associated with an object, or judgment of motion direction from optic flow signals. Some regions of the visual cortex are activated more strongly when the visual perception contain moving elements. Another interesting aspect of the human motion processing ability is the perception of the biological motion in which humans can differentiate states such as running and jumping and even gender of the actor from poor visual scenes in which only a few light-points are included. Similar to motion processing, depth feeling entail both low-level visual features such as disparity and high-level inferred attributes such as the surfaces corresponding to retinal points with different disparities.

Nevertheless, the main secret of the human visual perception is the objects recognition ability with high accuracy and speed in presence of drastic changes in the appearance of objects caused by changes in the viewing angle and illumination conditions. Visual cortex object-selective regions are activated when subjects view objects defined by luminance, texture, motion or stereo cues but not when subjects view textures, stationary dot patterns, coherently moving dots, or gratings defined by either motion or stereo while activation is independent of object format or scale. In general, object-selective regions represent shapes rather than contours or object features when subjects perceived simple shapes created via illusory contours. These properties form an object-recognition system with a wide range of invariances to the object appearance and transformations but not its identity.

Two main principles of the visual cortex functionality are the hierarchical processing and

the functional specialization. During hierarchical processing, the visual perception is achieved by a gradual stage-wise process in which information is first represented in a localized and simple form that will be transformed into more abstract representations after a sequence of processes. The functional specialization proposes that specialized neural pathways process information about different aspects of the visual scene. It is almost approved that the human visual system is consist of parallel hierarchical processing streams that each of them is specialized for a particular task.

1.3 Machine Vision

Photos and videos are becoming an integral part of the global life. They are being generated at a pace that is far beyond what any human or teams of humans could hope to view. But unfortunately, our most advanced software is still struggling with understanding and managing this enormous content. We have made fabulous megapixel cameras that are able to take pictures by converting lights into a 2D array of numbers known as pixels but these are just lifeless numbers. They do not carry meaning in themselves. Prototype cars can derive by themselves, but without *smart vision*, they cannot really tell the difference between a crumble paper bag on the road, which can be run over, and a rock in the same size which should be avoided. Drones can fly over massive land, but do not have enough vision technology to help us to track the changes of the rain-forests. Security cameras are everywhere but they do not alert us when a child is drowning in a swimming pool. So, collectively as a society, we are very much blind because our smartest machines are still blind.

Just like to hear is not the same as to listen, to take pictures is not the same as to see and by seeing we really mean understanding. In fact, it took the mother of nature 540 million years of hard work to do this task, and much of that effort went into developing the visual processing apparatus of our brain, not the eyes themselves. So, vision begins with the eyes but it truly takes place in the brain.

Visual intelligence is a cornerstone of the Artificial Intelligence (AI), as the computer must understand what it sees, and then perform appropriate reasoning and actions accordingly.

So, the ultimate goal of the computer vision is to enable machines to see, interpret and process visual data in the same way that human vision does. In other words, it can be defined as the transmission of the human intelligence and instincts to a computer. Naming objects, identifying people, inferring 3D geometry of things, understanding relations, emotions, actions, and intentions. But it is quite a challenging task to enable computers to differentiate between several kinds of visual information such as shape, object, and texture as well as certain 3D vision interpretations such as depth, illumination, and motion.

It used to be that if we wanted to get a computer to do something new, we would have to program it. Programming is a process that requires pondering about every single step in order to achieve the final goal. But no one tells children how to see especially in the early years. They learn this through real-world experiences and examples. If you consider a child's eyes as a pair of biological cameras, they take one picture about every 200 milliseconds which is the average time an eye movement is made. So, by the age three, a child would have seen hundreds of millions of pictures of the real world which is a lot of training examples.

Therefore, maybe the better solution for computer vision problems is to construct algorithms that are able to learn from and make a prediction on data. This smart programming approach that is called *machine learning* is a cumulative knowledge acquiring technique that uses iterative algorithms and statistical analysis to recognize trends from data with the aim of finding hidden insights without being explicitly programmed.

Most of the techniques in machine learning such as neural networks, support vector machines, decision trees, Bayesian belief networks, k-nearest neighbors, self-organizing maps, case-based reasoning, instance-based learning and hidden Markov models are not new but today the abundance of the data and the affordable processing power beside inexpensive data storage facilities make it possible to deliver faster and more accurate outcomes based on larger scale and more complex data which means better decisions for financial services, health-care, transportation, marketing and much more in real-time without human intervention.

A fundamental step towards the progress of the computer vision is to teach computers to watch *objects* that form main building blocks of the visual world. *Object recognition* in

turn is a very challenging task. Even something as simple as a household pet can present an infinite number of variations to the object model. Therefore, instead of focusing solely on better and better algorithms, the solution might be to give the algorithms the kind of training data that a child is given through real-life experiences in both quantity and quality. The prerequisite of such a task is the access to the abundant environmental information consist of meaningful properties and relations. Consequently, we need to collect different data sets that have far more images than we have ever had before, perhaps thousands of time more. Internet as the biggest treasure of images that humans have ever created can be used to form those required training sets.

When we have the data to nourish the computer brain, then it is time to come back to the algorithms themselves. The wealth of information provided by large-scale data sets is a perfect match to a particular class of machine learning algorithms called Convolution Neural Networks (CNNs). Just like the brain, a CNN consists of billions of highly connected neurons. A neuron is a basic operating unit in a neural network that takes input from other nodes and sends output to others. In addition, these hundreds of thousands or even millions of nodes are organized in hierarchical layers also similar to the brain. A typical neural network has about 20 million nodes, 140 million parameters and 15 billion connections that form an enormous model. Powered by the massive data of the newly provided visual data sets and the modern Graphics Processing Units (GPUs), the CNN became a winning architecture that generates exciting new results in various machine vision tasks.

After understanding the surrounding world, a small child learns the natural language to exchange the knowledge, interact with environment and exploit the collective consciousness. As the primary step to teach a computer to see a picture and generate sentences, the combination of the big data technology and machine learning algorithms has to take another step in such a way that the computer can learn from both pictures and their natural language descriptions generated by humans. Just like the brain that integrates vision and language, the main contribution of this thesis is to develop models that connect image regions as visual snippets with their linguistic interpretations in the form of words, phrases and sentences.

When we meet our dreams, doctors and nurses will have extra pairs of tireless eyes to help them to diagnose and take care of patients, cars will run smarter and safer on the road and

robots will help us to brave the disaster zones to save the trapped and wounded. We will discover new species, better materials and explore unseen frontiers with the help of the machines. First, we teach computers to see and then they help us to see better. For the first time, human eyes will not be the only ones pondering and exploring our world.

1.4 Outline of the Dissertation

In this dissertation, we develop models to bridge the gap between the visual data and its linguistic description. In particular, we develop deep architectures that process and align these two concepts and train their parameters end-to-end using data sets that are including image regions and their corresponding lingual explanations.

The literature review of the deep learning technology is presented in *Chapter 2* where we provide an introduction to different learning approaches, artificial neural networks, back-propagation mechanism, optimization approaches, overfitting, and regularization, and describe commonly used deep neural architectures for processing images and texts, especially convolutional and recurrent neural networks.

In *Chapter 3*, we develop a hybrid interactive model that is able to not only generate a wide range of linguistic descriptions for different regions of the input image but also detect user attention, highlight the intended region and provide the most appropriate linguistic description for its visual content.

In *Chapter 4*, we specifically address the problem of *selective texture labeling* as a bi-directional correlation between texture visual attributes and their lingual descriptions. The proposed model is able to receive visual textures as inputs and generate informative linguistic descriptions. The bi-directional nature of the proposed architecture which is formed by the combination of convolutional and recurrent neural networks also provides this opportunity to retrieve texture images based on linguistic descriptions of the intended texture attributes. As a prerequisite, we generate a multi-label data set of texture images by combining two well-known texture data sets, each of which provides a set of texture visual attributes in a unique context.

In the both previously proposed deep architectures, morphological operations are exten-

sively used to speed up the pre-processing steps of our colored training samples. Motivated by such an application, in *Chapter 5*, we provides a novel approach to define mathematical morphology on matrix field data. To this end, we converted common RGB-values of the images into a new biconal color space and then combined two approaches of mathematical morphology to give meaning to the maximum and minimum of the matrix field data and formulate our novel strategy.

Chapter 2

Deep Learning Background

In the early days of artificial intelligence, smart solutions were limited to problems that their description was a symbolic representation of the domain and the manipulation of symbols based on the given rules. In that context, an intelligent agent was defined as an efficient search engine that was able to traverse a limited solution space better, faster and more greedy than a human being. One famous example of such a task, is the chess game and the ability of the artificial agents to examine the movement possibilities and predict their consequences. The scope of such a solution was relatively confined to problems that could be easily formulated. This was much lower than the area of the human intelligence where the agent is able to leverage a large amount of knowledge about the problem gathering from linguistic communications, visual interpretations and other kinds of knowledge resources. Humans unconsciously evaluate and discard many options that are not related to the subject and make a choice from millions of options whenever they act.

To be able to reach that level of intelligence, knowledge-based systems were presented wherein the knowledge was prepared by specialists and expressed in a way which allowed the agent to reason about different situations. Machines are not able to understand the actual meaning of the knowledge since they are not able to understand the concept that put knowledge together. So, the main limitation at that time was the need of the external knowledge interpretation. In that sense, one had to define every possible value that might be needed for the task known as *features*. Humans are good at recognition and construction of these features. We are familiar with this process as *learning*.

Those days, the *digital learning process* was defined as the development of algorithms that were able to obtain relevant knowledge of the problem by proper mapping of the data provided by human experts. So, presentation of the most accurate mathematical function that describes internal relations of the data was the masterpiece of the field of machine learning. Thanks to the diversity of machine learning techniques, a large class of problems were solvable by providing enough data, engineered features, and computational power. But one has to notice that machine learning algorithms are able to produce an appropriate answer to the question only when some relevant features are provided. Unfortunately, machines were not able to judge what a suitable attribute is, so the main obstacle of these approaches is the need of the breathtaking *feature engineering* especially in the domains of new problems. Based on that, the meaning of the intelligence was the identification of a feature combination that yields highest precision.

Human learning goes from raw data to a conclusion without the explicit step where features are identified and provided to the learner. In other words, we learn the suitable representation of the data from the data itself. In addition, our brain is able to construct a hierarchy of features in which complex concepts are presented by primitive elements known as *automated feature engineering*. The hierarchical structure of the artificial neural networks is designed to partially simulate this natural property. In general, they consist of simple, highly interconnected processing elements or neurons with weighted interconnections among them. Neurons (a.k.a. nodes) are arranged in multiple layers, including an input layer where the data is fed into the system, an output layer where the answer is given, and one or more hidden layers, where the learning takes place by updating the weights of the interconnections.

Deep learning, in turn, refers to neural networks that include a large system of neurons arranged in several layers. In such networks, each layer of nodes trains on a distinct set of features based on the previous layer's output. With further advancement through the network, neurons aggregate and recombine features from the previous layers, so they are able to recognize a hierarchy of increasing complexity and abstraction. This important property makes deep networks capable of handling very large, high-dimensional data sets with billions of parameters that pass through nonlinear functions.

One of the problems deep learning solves best is the processing of the raw (unlabeled) data and discerning similarities and anomalies that no human has organized in a relational database or ever put a name to. This covers sound, text, images, videos and time-series-shaped information such as web activities. Deep learning can be seen as a system that learns a set of intermediate representations for specific tasks. Developing optimal intermediate feature representation for images or videos is the crucial step for the computer vision and multimedia applications. The more data a network can train on, the more accurate it is likely to be in the prediction of the outcome related to unseen but similar situations. This is a new information processing method which has led to ground-breaking achievements in visual recognition, natural language processing and robotics.

Deep learning platform provides an amazing opportunity to think of some really fascinating applications including but not limited to automatic image colorization, adding sounds to silent movies, image style transformation, machine translation, voice recognition, fraud detection, object detection, text and handwriting generation, sentiment analysis, game playing and automatic scene description.

2.1 Learning Approaches

Learning in its broad interpretation is the process of earning the ability to perform a task. There is a wide variety of the tasks that can be done by a learning approach. Machine learning in turn, can be divided into three broad groups of algorithms: supervised learning, reinforcement learning, and unsupervised learning.

2.1.1 Supervised Learning

While the aim of the traditional algorithms is to provide a closed-form solution for the given problem, supervised learning approaches try to find the hidden relation between input and the output of the samples representing the problem space. Supervised learning itself has two different branches. In *regression*, the desired output is a real number or a vector of real numbers and the aim is to get as close as we can to the correct output value. In *classification*, the requested output is a class label (categorical value) which can be in

the simplest case a choice between one and zero or positive and negative. Obviously, it can include more complicated situations such as having multiple alternative labels in the task of visual recognition. As a visual classification, an image can be fed to the algorithm and the output is a set of the probabilities that determine the presence of the predefined objects in the image.

The first step of the supervised learning is the selection of the model class from the whole set of models that were prepared to consider as candidates. The model class can be considered as a function that takes an input vector X and some numerical parameters θ and gives you an output Y . The typical input vector includes several quantitative measures or attributes. The aim is to adjust the set of parameters to make the mapping fit with the supervised training data. Here the word "fit" can be described as minimizing the discrepancy between the target output at each training case and the actual output generated by the model.

A common measure of that discrepancy is the *mean squared difference* between the vector of the system output Y and the ground truth vector \hat{Y} . Normally we put in that half so it cancels the two when we differentiate:

$$L = \frac{1}{2N} \sum_{i=1}^N \|\hat{Y}_i - Y_i\|^2 \quad (2.1)$$

Here N is the number of data samples. This measure is more sensible when we tackle a regression problem.

2.1.2 Unsupervised Learning

For many years machine learning scientists had neglected the power of unsupervised learning and had confined its application to some sorts of clustering approaches. Consequently, they were used to define machine learning algorithms as some mapping functions from input into output. The possible reason for this situation was the dilemma to explain the aim of unsupervised learning which is to create an internal representation of the input data that is useful for subsequent supervised or reinforcement learning. Another goal of the unsupervised learning is to transform a high dimensional input into a compact low dimensional

representation and it comes from this fact that high dimensional data typically lives on or near a low dimensional manifold [108]. An almost simple method for *dimensionality reduction* is the Principal Component Analysis (PCA) that acts linearly. This means PCA assumes that there is only one manifold which is a $(k - 1)$ -dimensional hyperplane in the original k -dimensional space of the problem. The other aim of unsupervised learning is to supply an economical representation of the input in term of learned features. For example, if we can represent the input in terms of binary features, that representation is economical while it only takes one bit to express the state of a binary feature. Another shape of the economic representation of an input is to use a large number of real-valued features that are mostly zero which is known as *sparse representation*. In this case, for each input, we only need to represent a few real numbers. As the most common application of unsupervised learning, clustering can be defined as a very sparse code in which we have one feature per cluster and we force all the features except one to be zero while that specific feature has the value of one. Hence, the clustering is an extreme case of finding sparse features.

2.1.3 Reinforcement Learning

Reinforcement learning is another type of machine learning algorithms where an agent learn how to behave in a environment by performing actions and seeing the results. The agent receives rewards by performing correctly and penalties for performing incorrectly. So, the agent aims to select actions or sequences of actions to maximize the expected sum of future rewards while rewards may occur occasionally. Normally a *discount factor* is used to limit the search domain in the upcoming possibilities. In a simple definition, the discount factor decreases the reward for the actions as far as we dive deeper into the future. Since rewards are mostly delayed, it is hard to know which action is the wrong one in a long sequence of actions. One more difficulty of the reinforcement learning comes from this fact that scalar rewards especially those occur occasionally do not provide much information on which to base the changes in parameters. Therefore, in contrast with supervised and unsupervised learning, it is not possible to use reinforcement learning to learn more than thousands of parameters.

2.2 Artificial Neural Networks

Humans are easily able to accomplish sophisticated visual recognition exploiting different cortices of the brain. This admirable ability is obtained during a long-term natural evolution in a world where the electromagnetic waves are the only source of the knowledge. The human brain is an extremely complex combination of neurons in which each individual neuron can form thousands of links with other neurons. Converting such a capability to the digital world is a tough mission.

The primary phase of this process is done in 1950s with the definition of the Artificial Neural Networks [167]. Inspired by the brain structure, the fundamental building blocks of the neural networks are independent computing elements known as neurons or nodes. As illustrated in Fig. 2-1, a biological neuron consists of several axon terminals connected via synapses to dendrites on other neurons. When the brain learns a new task or forms memories, it encodes each piece of the new knowledge by tuning connections between neurons. However, the contribution of a single neuron to computation in the brain should not be underestimated. Neuronal dendrites of each neuron exhibit a range of linear and nonlinear mechanisms that allow them to implement elementary computations [130].

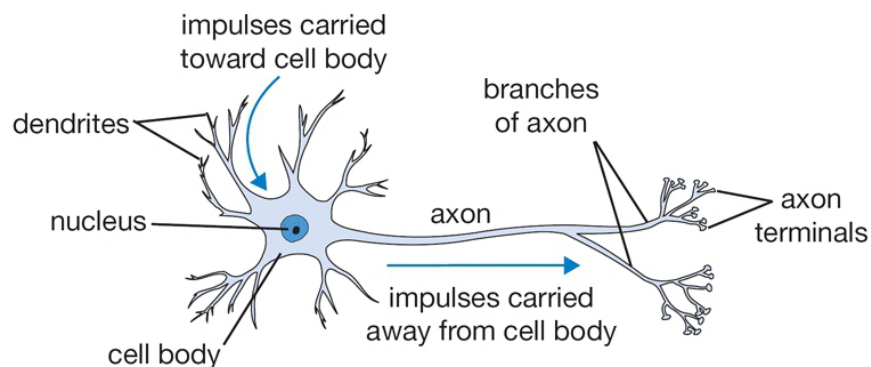


Figure 2-1: The general structure of a biological neuron [3]. The figure is redrawn based on [4]

If the aggregation of the incoming signals reaches a specific threshold, the neuron fires the response along the axon. Digital version of a neuron (a.k.a. perceptron) takes several inputs, determines each input impact by its coefficient (weight), computes their summation

and sets the output in the case of the threshold satisfaction. As the basic mathematical aspect of the neural networks is the function estimation, applying a naive thresholding (e.g. an step function) on the linear combination of inputs preserves the linearity while lots of interesting functions have nonlinear properties. As the result, in current network architectures, a nonlinear function such as sigmoid or hyperbolic tangent is used as an activation function to squash the output range of the neuron into a certain interval and equip networks with nonlinear functionality. In addition, there can be an extra variable (bias) in each neuron that affects the initial level of squashing which results in the desired non-linearity. The basic structure of an artificial neuron is shown in Fig. 2-2.

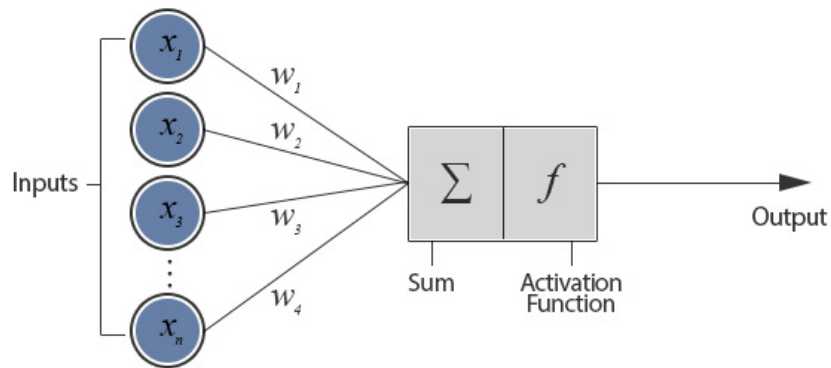


Figure 2-2: The general structure of an artificial neuron [3].

Synaptic coefficients (weights) can be changed to affect the influence of each neuron on another. This mechanism is known as the *learning process* in neural structures which determines the dynamic of the model performance.

2.2.1 Feed-Forward Networks

Neural Networks are basically organized into distinct sets of nodes called layers. This structure makes it very convenient to analyze their operation using matrix multiplication. Early attempts that tried to simulate a naive version of the neural system have relied on *feed-forward* models. This structure consists of neurons that are connected in an acyclic manner to arrange the output of each layer as the input of the neurons in the next layer. The most common layer type is the *Fully-Connected* (FC) layer in which neurons between two

neighboring layers are fully connected, but neurons within a single layer are not directly connected to each other. Nodes of the input layer have no parameters and do not modify the data. In contrast, nodes of intermediate (hidden) and output layers carry parameters, perform the learning approach and are counted in the naming convention. Output layer mostly represents class scores and real-valued numbers in classification and regression. Consequently, output neurons are not necessarily equipped with activation functions. It turns out that networks with even one hidden layer are universal approximators [89]. With the rise of the size and the number of layers, neurons can collaborate to express more complex functions which grow the space of representable functions and increases network capacity. The potential risk of this situation is the *overfitting* that occurs when the network employs its high capacity to fit all the observations in the data that may also include noises and outliers. Therefore, the higher model capacity must be appropriately addressed with stronger *regularization*.

2.2.2 Output Layers and Loss Functions

Softmax Output layer

Softmax function is a kind of soft continuous version of the maximum function that is used to force the output of the neural network to sum to one so that it can represent a probability distribution across discrete mutually exclusive alternatives. In a softmax output layer, each node receives some total input that they have accumulated from the previous layer. Then, the i th neuron generates an output y_i that does not only depend on its own input z_i but also on the inputs of other neurons z_j of the softmax group as well. So, the output of the i th neuron is:

$$y_i = \frac{e^{z_i}}{\sum_{j \in \text{group}} e^{z_j}} \quad (2.2)$$

Since the denominator of the equation is the sum of the numerator over all possibilities, the sum of outputs for all the neurons in the softmax output layer will be summed in one and each output y_i lies between zero and one. Consequently, network outputs represent a probability distribution over mutually exclusive alternatives. The other suitable property of

the softmax equation is its simple derivative:

$$\frac{\partial y_i}{\partial z_i} = y_i(1 - y_i) \quad (2.3)$$

Where the derivative of the output with respect to the input for each neuron of the softmax layer depends only on the output itself. In the case that we decide to use the softmax output layer in a network, we should think of the right loss function and that is the negative log probability of the right answer known as the *cross-entropy* loss function:

$$C = - \sum_j t_j \log y_j \quad (2.4)$$

Which means we aim to maximize the log probability of the right answer. Here j iterates over all the neurons of the softmax group, while y_j and t_j are the actual and the target outputs of the neuron. One should notice that it is necessary to add up across all the neurons of the layer because when one neuron changes, outputs of the other neurons change as well. This loss function has a very steep derivative when the answer is very wrong which balances the flatness of the derivative of the error with respect to the changes in each layer input:

$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i \quad (2.5)$$

So, when the actual and the target output of a neuron are very different, the aforementioned derivative has the value of 1 or -1 which indicates the maximum possible slope.

2.2.3 Training Terminology

Forward-pass and *backward-pass* are the essential units of the network computations. The forward-pass computes the output given the input for inference. The backward-pass computes the gradient given the loss of learning. In addition, an *epoch* is a complete pass through the training data. At the time of one epoch, the network meets every training sample once. Normally, the whole set of training samples is too large to be fed entirely to the network during one forward-pass. So, the solution is to divide training samples into

smaller groups or *mini-batches*, feed the network with one mini-batch and update network parameters afterward. This hybrid training step is called an *iteration*.

2.2.4 Depth Influence

While the universal approximation property holds both for deep and shallow networks, by adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. In addition, some of the basic visual elements that are necessary to perform a successful recognition task including corners, junctions, contours, and figures can be defined as a spatial correlation between some points. So, an image can be considered as a hierarchical structure and several layers of processing make intuitive sense for this data domain. During the training process of a shallow network, weights are properly allocated from the learning data and the network can recognize and classify patterns well. However, once the numbers of the network layers increased, the linkage becomes too dense and it is difficult to make a difference in the weights. The network adoption with the data is made by a mechanism that feeds back errors that occurred during training to the whole network. For a network with several layers, the error disappears before it is reflected the whole network which is known as the vanishing gradients problem. In 2006, Hinton and Salakhutdinov [83] introduced a learning approach that effectively trains deep architectures one layer at a time in which the data that is learned in the lower layer is treated as input data for the next layer. As each layer learns in stages, the feedback for an error of learning can also be done properly in each layer. This leads to an improvement in precision.

2.3 Backpropagation

In this section, we overview the concept of learning hidden nodes of a neural network known as *backpropagation* algorithm. Generally, if we do not exploit hidden neurons in neural networks, they mapping power between input and output decrease dramatically. In that situation, adding perceptrons containing hand-coded features in between is able to improve network capacity while the design of the features remains an obstacle. What we prefer is a way to find good features without requiring insights to the task or repeating trial

and error to guess some features and see how well they work.

Our aim is to automate the loop of the feature construction and its evaluation. The naive way of doing so is to randomly perturb weights, check the system performance and save those weights in the case of improvements which is the typical action in reinforcement learning. The main problem with this approach is its inefficiency. To change only one weight, we need to perform several forward passes on a representative set of training cases and check the results. Furthermore, as towards the end of the learning process, weights have to have relative values to work properly, any large change in a weight will nearly always make things worse. There are slightly better ways of using perturbations in the learning process. One suggestion is to perturb all the weights in parallel and then correlate the performance gain with the change of the weights. Actually, this technique does not help so much while we need to do a huge number of trials with different random perturbations of all the weights in order to observe the effect of changing one weight through all the noise that is created by changing all the other weights. In comparison with this technique, backpropagation is a much more efficient by a factor of the number of weights that could be millions.

Another way to improve learning process is to randomly perturb the activities (outputs) of the hidden units instead of perturbing the weights. Once we know how we want a hidden activity to change on a given training case, we can compute how to change the weight. As there are many fewer activities than weights, there are fewer possibilities for random exploration which makes the algorithm more efficient. In comparison with this modified strategy, backpropagation is still more efficient by the factor of the number of nodes.

The fundamental logic of the backpropagation is that although we are able to compute how fast the error changes as we modify the hidden activity, we are not aware of the correct state of the hidden units. So, instead of using activities of the units as the desired states, we are able to use *error derivatives* with respect to activities for the learning. Since each hidden neuron affects several output units, it has many different effects on the total error that should be combined. This computation allows us to efficiently compute error derivatives for all the hidden units at the same time. The result of having those derivatives is we are able to find out how fast the error changes as we change the hidden activity on each particular

training case. It is easy to use the *chain rule* to compute error derivatives of the weights coming into a hidden unit from the error derivatives of the activities. As the first step of the backpropagation, we have to define the error which is in a simple case the squared error between the target value of the output unit \hat{y}_j and the actual value that the network produces for the output unit y_j :

$$E = \frac{1}{2} \sum_{j \in \text{output}} (\hat{y}_j - y_j)^2 \quad (2.6)$$

The next step is to differentiate the error with respect to the activity (output) of an output unit:

$$\frac{\partial E}{\partial y_j} = -(\hat{y}_j - y_j) \quad (2.7)$$

Once we have got the error derivative with respect to one of the output units, we then want to use all those derivatives in the output layer to compute the same quantity in the hidden layer that comes before the output layer. So, the core idea of the backpropagation algorithm is to take error derivatives in one layer and compute the error derivatives of the previous layer based on that. Obviously, when we change the output of a unit in a specific layer, it will change outputs of its subsequent units in the next layer and this is the situation that we should sum up all these effects. The plan is to have an algorithm that takes error derivatives that we have computed for the top layer and combines them in the same manner as we used in the forward pass to calculate error derivatives in the layer below. Fig. 2-3. Shows a multi-layer architecture in which the output unit is called j and the hidden unit is called i . The output of each unit is denoted by y including unit index. The total or weighted input received by j which is denoted by z_j is simply the multiplication of the output of the hidden layer y_i and the weight between two units w_{ij} :

$$y_j = \delta(z_j) = \delta(w_{ij} \cdot y_i) \quad (2.8)$$

The first thing we need to do is to convert the error derivative with respect to y_j into an

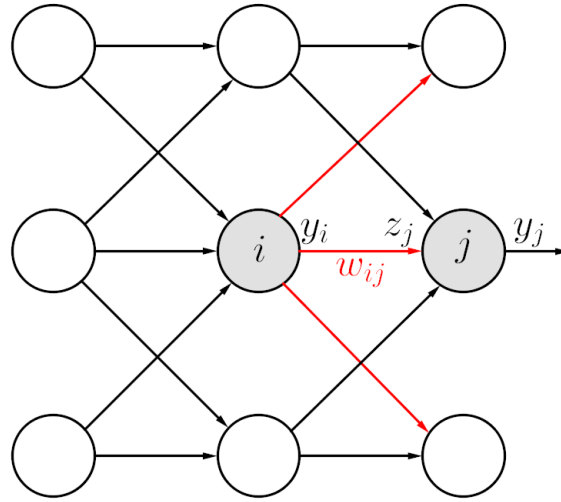


Figure 2-3: Error derivatives calculation by backpropagation [28]

error derivative with respect to z_j . To do so, we can use the chain rule as follow:

$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j} \times \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j} \quad (2.9)$$

So, now we obtained the error derivative with respect to weighted input of the output unit. Now, it is time to compute the error derivative with respect to the output of the hidden unit:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i} \times \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} \quad (2.10)$$

Which is computed by the sum over all outgoing connections of the hidden unit. In the first term of the chain in this expression, we explain how weighted input of the output unit z_j changes as we change the output of hidden unit y_i which obviously equals to the weight on the connection w_{ij} . Once we have got the error derivative with respect to the output of the hidden unit, it is very easy to compute the error derivatives for all the outgoing weights of the hidden unit:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \times \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j} \quad (2.11)$$

We can clearly do that for as many layers as we like and after we have done that for all existing layers, we can compute how the error changes as we change the weights of the network. In spite of its useful properties, backpropagation is not a learning algorithm by

itself. For a proper learning procedure, we need to have a specific strategy to tune all the effective parameters. The main point is the approach that we employ to optimize the error function or the way that we use error derivatives to discover a good set of weights.

Another concern is the generalization strength of the weights that we have learned. In other words, we have to be sure that our weights are able to handle input samples that are not seen during training.

2.4 Optimization

The error surface is the surface that lies in the problem space where the horizontal axis corresponds to the weights of the network and the vertical axis corresponds to network error rate. For a linear neuron that has a squared error, the error surface is a quadratic bowl where the vertical cross-sections are parabolas and horizontal cross-sections are ellipses. In contrast, for a multi-layer, non-linear neural network, the error surface has a very complicated geometry. Fortunately, these complex error surfaces are also locally quadratic.

2.4.1 Gradient Descend

The major target of the optimization process is to determine how to use weight derivatives. To clarify the answer to this question we need to know how often should we update the weights. One possible strategy is the *online learning* in which we make small changes in weights after meeting each training sample and computing of error derivatives with respect to that sample. This strategy leads to severe fluctuations in the amount of the error function due to different derivatives that we obtain along each observation. Nevertheless, in such a case, the whole optimization procedure is able to converge towards minimum if we apply tiny changes in each step. The other feasible approach is to use full batch or *offline training*. To that end, we perform a full sweep through all the training samples, sum up all the error derivatives that we earn during whole observations and take a small step in that direction. Note that this learning strategy keeps the weights constant during the computation of the error associated with each sample while the online version is constantly updating the weights, so the two algorithms visit different sets of points during adaptation. However,

they both converge to the same minimum. The offline learning is slightly more efficient in terms of the number of computations. The problem with the offline learning is that if we start with a set of weights that includes a few inappropriate values, it is not necessary to look at the whole bunch of the training set to get the idea of which direction we need to move. In such a case, a few training samples are sufficient to lead us in the right direction. The third practical method of training is *mini-batch learning* where, in each iteration, we select a small random group of training samples to feed the network, perform backpropagation and compute error derivatives. Then the network weights will be slightly changed with respect to error derivatives. Consequently, we face less variation in the network output in comparison with online learning, while benefiting from the efficiency of the batch learning as well. While the gradient leads us towards the direction in which the function has the steepest rate of increase/decrease, the next issue, is the amount of the change that we want to make in the weight updates. This amount is determined by a coefficient (η) that we use before the amount of the error derivative:

$$w_t = w_{t-1} - \eta \frac{\partial E}{\partial w} \quad (2.12)$$

This coefficient is a learning hyperparameter that we call step size or the *learning rate*. The simplest strategy is to use a fixed value for the learning rate during the training. But it seems more sensible to adopt the learning rate based on the overall trend of the network accuracy improvement. In that sense, we should decrease the learning rate when the network fluctuates or stops learning any more. In contrast, the learning rate can be increased when the learning progress is steady. It is also plausible to have a diverse set of learning rates for different parts of the network so that some parts learn rapidly and other parts learn slowly. Moreover, in some cases, the best learning strategy is not to traverse the problem space exactly in the direction of the steepest descent. For example, as illustrated in Fig. 2-4, in some situations where the problem space is highly elongated, the direction of the steepest descent may not conform the direction of the minimum point that we aim to go through. In other words, the gradient is high in the direction that we prefer small steps and the gradient is very low in the direction that we wish to take long steps. But the main challenge is still

to find a way to choose such an efficient movement strategy.

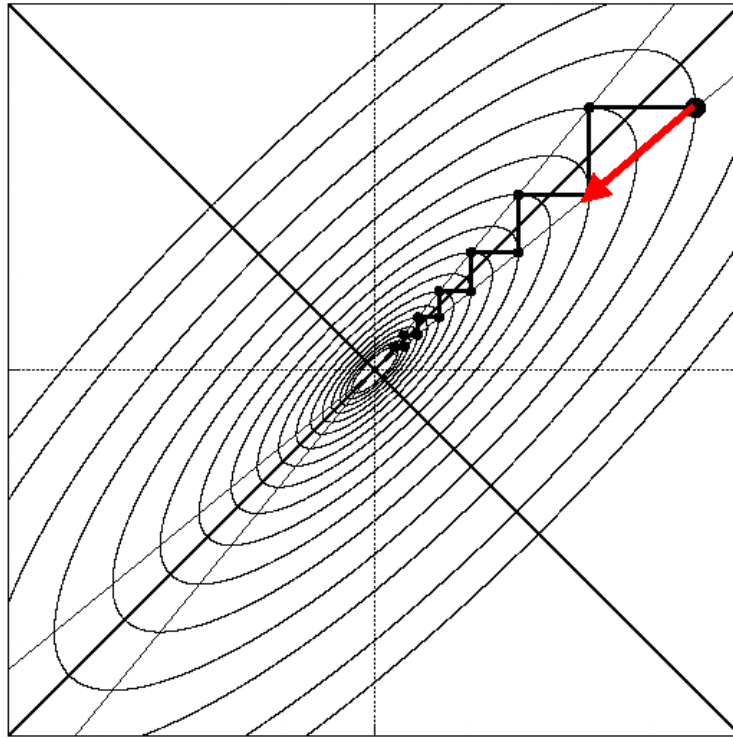


Figure 2-4: A demonstration of an elongated problem space. The figure is redrawn based on Figure 2.1 in [104]

2.4.2 Some Optimization Catalysts

In this section, we will describe some tricks that are able to accelerate optimization process across a multi-layer neural network. As we use a numerical optimization routine, if the weights of the network have exactly the same initial parameters, they will generate exactly the same gradient. The outcome of such a situation does not lead to automatic generation of different features. So, the best way to break network symmetry is to initialize network parameters with *small random values*.

Another strategy which can improve learning quality is to transform each component of the input data so that it has *zero mean* and *unit variance* over the whole training set. The geometrical interpretation of such a technique is to centralize the data point cloud symmetrically around the origin for every dimension. One possible way for zero centering is to use hyperbolic tangent activation for all neurons. This transformation can also be done by

mean subtraction technique during data preprocessing step.

The other approach that grants the circular shape of the error surface is the decorrelation of the input vector components. PCA approach is able to perform such a decorrelation by removing components that have small eigenvalues. Afterwards, dividing remaining components by the square roots of their eigenvalues modifies the elliptical shape of the error surface to a circular shape in which gradients straight towards the minimum.

There is another trick to speed up the learning process. Normally, we start the procedure by guessing the initial learning rate. Then, we monitor the learning progress and reduce the learning rate if the error rate gets worth or oscillates wildly. It also sounds reasonable to increase the learning rate if the error is falling too slowly. In the special case of learning with mini batches, towards the end of the learning process, we get strong fluctuations in gradients that cause the severe changes in the network parameters. So, it always helps if we turn down the learning rate at that moment to smooth fluctuations and get the final set of weights which forms a good compromise. A good approach for network monitoring is to check the network accuracy dynamic on a separate *validation set*. Since turning down the learning rate also reduces the rate of the learning, we have to pay careful attention to the time of the decrease.

2.5 Overfitting and Regularization

As the main source of the overfitting problem, training data can be easily ruined by the different type of noises. The minor type of the noise comes from *erroneous data labels*. But the most important sort of the noise is the *sampling error* which can be seen in the shape of random regularities that are caused by some particular samples that were chosen by mistake. As a rule of thumb, no matter how accurate the training data is, for any finite set of examples, there is always some accidental regularities. The main challenge is that there is no way to distinguish actual regularities from those that are caused by sampling error. Unfortunately, all the models tend to fit both kinds of regularities. Obviously, more powerful models fit the data better and can be easily trapped by fitting the sampling error. So, overfitting occurs when the network predicts the training set very well but makes poor

predictions on unseen data.

Although overfitting is the main obstacle of the network generalization, it can be cured by several *regularization* techniques. One way of improving network generalization is to prevent network weights to be too large also known as *weight decay* technique. This can be done by adding a *regularization term* to the loss function. The most common regularization terms are $L1$ and $L2$ norms of the weights.

Another technique for the network generalization improvement is the *weight sharing* approach where we try to give same values to many weights. This value should be learned during the training phase.

Early-stopping strategy combats overfitting by monitoring network performance on a validation set that is a set of examples we never use for gradient descent, but which is also not a part of the test set. If the network accuracy ceases to improve sufficiently on the validation set, or even degrades with further optimization, then the heuristic implemented here gives up on much further optimization.

By model averaging or *ensembling* technique, one can use several neural networks for the specific task and exploit the average output of those systems as the final output to diminish overfitting. These types of techniques are commonly applied on decision trees and Random forests.

There are other techniques to combine the output of multiple models to get better result. In *mixture* technique, this can be done by averaging models output probabilities. Another way of combining models is to use the *product* of the probabilities which is equivalent to their geometric mean. This geometric mean will generally add up to less than one, so we have divide by the sum of the geometric means to normalize the distribution. As a consequence, the small probability output by one model has veto power over the other models.

2.5.1 Dropout

Dropout technique is an efficient way to combine a large number of neural network models without having to separately train each individual model. To do so, we omit some of the hidden units for each training case to end up with a different architecture. In other words,

we use a different model for every training case by randomly turn off a fraction of nodes during each step of the training to stop too much cooperation between the hidden units and force the network learns multiple independent representations of the same data and improve generalization power of the network. Based on the dropout technique, a random connection layout is selected from possible 2^h layouts where h is the number of the hidden units. Obviously, all of this layouts share weights that is whenever we use a hidden unit, it gets the same weight as it has in other layout. The sharing of the weights between all the models means that each model is very strongly regularized by the others that leads to much better regularization than $L1$ and $L2$ penalties. Those penalties pull the weights towards zero, but by sharing weights between many models, each model gets regularized by something that pulls the weights towards the correct value.

In the test time, we can use all of the hidden units and halve their outgoing weights. In this way, we have the same expected effect as they did when we were sampling. This method computes the exact value of the geometric mean of the predictions of all the previously used 2^h models. When we have multiple hidden layers, this is not exactly the same as averaging all the separate dropped out models, but it is a fast and good approximation. The alternative solution is to run stochastic models several times on the same input and average across those stochastic models. This approach gives us an idea of the uncertainty in the answer which is an advantage over the mean net.

In general, if we have a deep neural network the overfits the data, dropout technique can reduce the number of the errors drastically. As a rule of thumb, any network that requires early stopping in order to prevent overfitting, would do better by using dropout with the cost of a longer training time and possibly more hidden units. In the case that our deep model does not overfit the data, that would be a good idea to use a bigger model that takes the advantage of the dropout regularization.

The other merit of the dropout regularization is that it prevents the *specialization* of the hidden units. This occurs when a hidden unit knows which other hidden units are present, so it can co-adapt to the other hidden units on the training data to compensate their possible mistakes. This may lead to *complex co-adaptations* [189] that does not generalize to unseen data and cause overfitting. The dropout technique is able to prevent co-adaptation by

making the presence of other hidden units unreliable. Hence, the hidden unit's functionality cannot depend on the performance of the other units.

2.5.2 Batch Normalization

Due to the modification of the weights during the training phase of a multi-layer neural network, the output distribution of each layer changes gradually that is mentioned in the literature as *covariate shift*. Therefore, top layers are forced to modify themselves with the shifts in incoming data distribution which yields to the reduction of the learning speed. As mentioned before, normalization as shifting inputs to zero mean and unit variance is often used as a preprocessing step to make the data comparable across features. Since normalization is a simple differentiable operation, it is possible to perform it all around the network rather than just performing it once in the beginning.

Batch Normalization (BN) technique [90] can be interpreted as doing normalization process at every layer of the network, but integrated into the network itself in a differentiable manner. Batch normalization performs the activation's normalization on each dimension of the input data independently, generally right before the non-linearity (activation function). During backpropagation, it uses dedicated learning parameters to adjust normalizing approach to the network learning progress. Furthermore, batch normalization simplifies the hyper-parameter tuning by widening the range of hyperparameters that the network can properly work with. In practice, networks that use batch normalization are significantly more robust to bad initialization.

2.6 Convolutional Architectures

In vanilla feed-forward networks, each node of a hidden layer is fully connected to all the neurons of the previous layer while neurons of an individual layer have no connections in between. Obviously, this fully-connected structure does not scale to larger images especially when dealing with three channel color images. For example, when we convert an ordinary color image with the common size of 256×256 into a one-dimensional input vector, this leads to $256 \times 256 \times 3 = 196,608$ input neurons. In deep architectures, we

would almost certainly want to have lots of neurons over multiple layers, so the number of the network connections would add up quite quickly.

A Convolutional Neural Network (CNN) has many fewer parameters than a fully connected network with the same number of hidden units. Such a network consists of several convolutional, pooling and non-linearity layers optionally followed by a few fully connected layers. The network input is normally a 3D matrix with $H \times W \times D$ dimensionality where H stands for Height, W stands for width and D stands for the depth of the input matrix.

As depicted in Fig. 2-5, in a conv/pooling building block, each neuron has selective connections to a small, local region of the preceding layer and weights of these connections form a filter (kernel) that will be evolved during the training phase of the network. At the time of the convolution process, each filter traverses the entire scope of the input matrix to generate its own *feature map*. In practice, filters often have the square shape of size 1×1 , 3×3 or 5×5 . Three hyperparameters control the size of the output volume: *depth*, *stride*, and *zero-padding*.

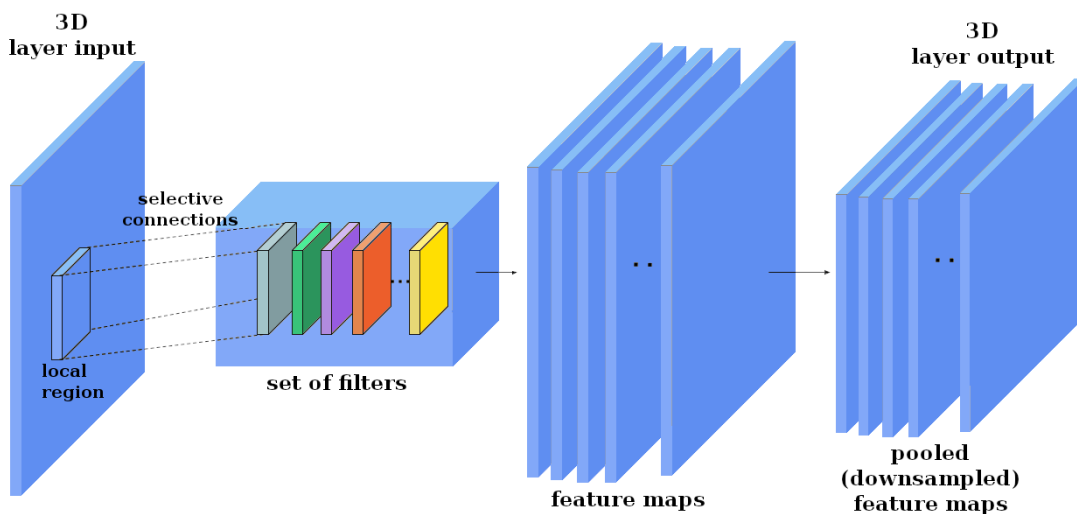


Figure 2-5: The general architecture of a conv/pooling building block in a CNN [28]

Depth corresponds to the number of filters in a convolutional layer, where each filter learns to look for something different in the input. For example, if the first convolutional layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. It is common to refer to a set of neurons that are all looking at the same region of the input as a *depth column*. The distance

between consecutive applications of a filter over the input scope is called *stride*. When the stride is n , we move the filters n pixel at a time. The bigger the stride, the smaller the output size. Sometimes it will be convenient to pad the input volume with zeros around the border. The size of this *zero-padding* operation enables us to control the spatial size of the output volumes. It also makes it possible to preserve the spatial size of the input volume, so, the input and output height and width remain unchanged. An important property of a convolution layer is that all spatial locations share the same convolution kernel, which greatly reduces the number of parameters of the layer. Because of sharing parameters, a kernel that is specialized to detect a pattern will detect it anywhere in the image.

As mentioned before, it is a convention to apply a non-linear function to grant non-linearity to a system that basically performs linear operations. In comparison with sigmoid and hyperbolic tangent activation functions, *Rectified Linear Unit* (ReLU) is able to greatly accelerate the convergence of stochastic gradient descent due to its linear, non-saturating form. Moreover, it can be implemented by simply thresholding the matrix of activations at zero. It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers. The ReLU layer applies the function $f(x) = \max(0, x)$ to all of the values in the input volume. In basic terms, this layer blocks all the negative activations.

Finally, to constantly reduce the dimension of feature maps, sharpen the located features, and control the amount of the computation in the network, it is common to perform a *pooling* operation after a convolutional layer. Typically, a pooling layer uses the maximum or the average operation to resize the dimension of the input and does not include any parameter. In *max pooling*, the pooling operator maps a sub-region to its maximum value, while the *average pooling* maps a sub-region to its average value. Furthermore, pooling helps to make the representation approximately invariant to small translation, rotation and shifting [71].

In this way, CNNs are able to encode a wide variety of visual features via their specific structure during the learning process. This kind of neural networks is usually used for visual classification, similarity detection (clustering), and object recognition.

2.6.1 LeNet

In 1986, a group of computer scientists showed that neural nets with many hidden layers could be effectively trained by a relatively simple procedure [171]. This would allow neural nets to get around the weakness of the perceptrons (incapability of learning the exclusive-or function c.f. [143]) because the additional layers endowed the network with the ability to learn nonlinear functions. Around the same time, the *universal approximation theorem* [86] showed that such networks have the ability to learn any function. Based on the universal approximation theorem, given any continuous function, no matter how complicated it might be, it is always possible to find a single layer feed-forward artificial neural network with a finite number of nodes which is able to approximate that function. In other words, by adjusting weights and biases, a neural network can take any input x and approximate it to fit the expected output $f(x)$. In addition, deep networks have a hierarchical structure which makes them particularly well adapted to learn the hierarchies of knowledge that seem to be useful in solving real-world problems. For example, in visual recognition tasks, deep networks provide a system that understands not just individual pixels, but also increasingly more complex concepts from edges to simple geometric shapes.

LeNet [115] is an excellent simple example for the application of the CNN architecture in the field of machine vision designed for handwritten and machine-printed character recognition. In this paper, authors proposed a gradient-based learning algorithm to synthesize the complex decision surface of the real-life document recognition systems. In their experiments, the goal was to classify the handwritten digits 0-9 and the model was trained on the MNIST [114] data set which is arguably the most well-studied data set in the computer vision literature.

As illustrated in Fig. 2-6, a sequence of conv/pooling blocks forms the lower part of the LeNet family of models. However, the upper part of the model consists of a fully-connected layer and the final classifier. In this way, the input to the fully-connected layer is the set of all the features maps of the layer below:

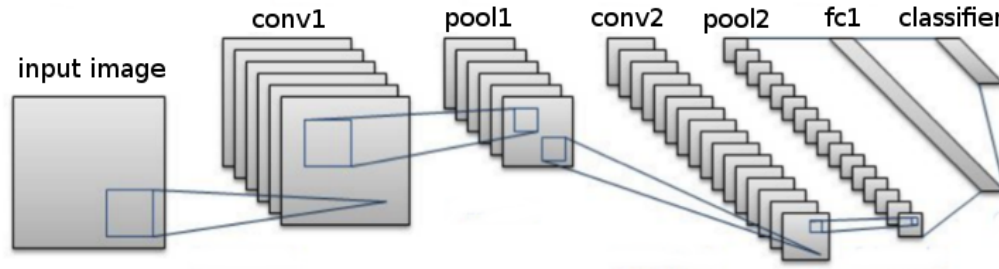


Figure 2-6: LeNet architecture [115], [3]

2.6.2 AlexNet

The neural network developed by Alex Krizhevsky and his colleagues known as AlexNet [107] was used to win the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [172]. This was the first time a model performed so well on a historically difficult ImageNet data set which contained over 15 million annotated images from a total of over 22,000 categories. The proposed network achieves the top 5 test error rate of 15.4%, while the next best entry achieved an error of 26.2%. This improvement was a breakthrough that totally amazed the computer vision society.

As it can be seen in Fig. 2-7, the network can be considered as the extended version of the LeNet architecture that was made of 5 convolutional and conv/pooling blocks, dropout layers, and 3 fully connected layers that perform classification task on 1000 possible categories. Beside the advantage of their new architecture that stacked hierarchical convolutional features on top of each other, further strengths of the AlexNet came from their wisely designed data augmentation procedure, application of the dropout layers for dealing with overfitting problem and ReLU activation functions that are less prone to be saturated, and their efficient CUDA [147] implementation to train the model on two separate GPUs.

2.6.3 ZFNet

The winner of the ILSVRC 2013 exploited an ensemble of architecture proposed by Matthew Zeiler and Rob Fergus [209]. In general, the new platform was an improvement on AlexNet by tweaking the architecture hyperparameters and achieved the error rate of 11.2%. To gain a better performance, they expanded the size of the middle convolutional layers and

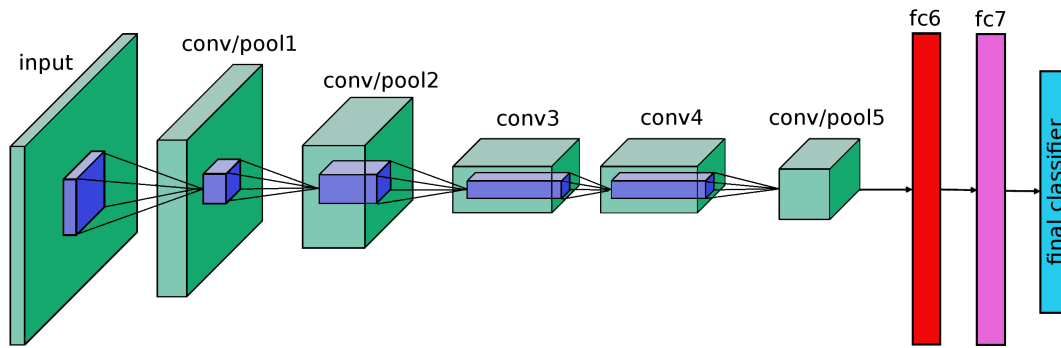


Figure 2-7: AlexNet architecture [107], [28]

decreased stride and filter size of the first layer. The reason behind such a modification is that a smaller filter size in the early convolutional layers preserves much more pixel information than the big filter size of 11×11 that is used in AlexNet. More importantly, authors introduced an interesting visualization technique named *deconvolution operation*, which is the transposed version of the convolution operation that helps to examine different features and their relation to the input space. To do so, a deconvolution layer will be attached to each layer of the trained CNN to provide an extra backward path to the input values. As in the normal forward-pass, activations will be computed at each level by feeding the input to the network. Then, for the examination of a certain node's feature map, the activation of the intended node will be frozen, while the outputs of all the other nodes in the layer are set to zero. During a specific backward-pass, the intended node provides the input for the deconvnet that has the same parameters as the original CNN. This input then goes through the series of deconvolution operations until the input space is reached. As the result, it is possible to determine what type of visual structures excite each node. The output of such a visualization for an early layer of the ZFNet is illustrated in Fig. 2-8. This visualization approach makes it possible to explain the inner functionality of CNNs as well as earning insights to design more robust network architectures.

2.6.4 VGGNet

The success of the VGGNet [184] proposed by the visual geometry group of the Oxford University in ILSVRC 2014, showed that the depth of the network is a crucial criterion for

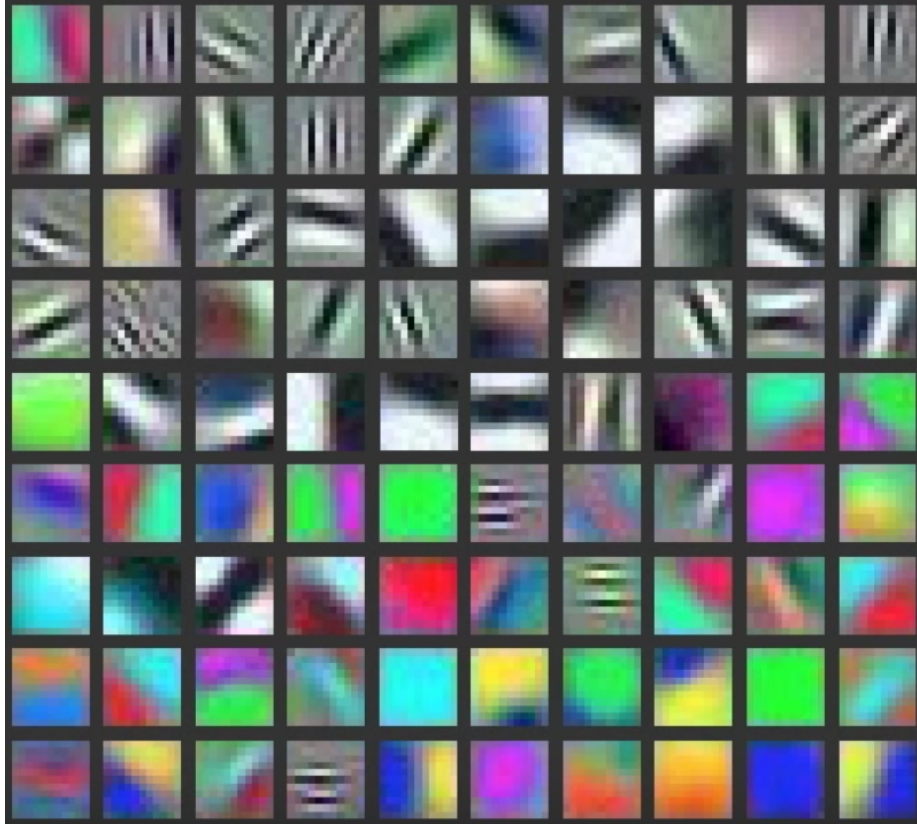


Figure 2-8: Visualization of a low-level convolutional layer as proposed in [209]. The figure is redrawn based on the figure 6 (c) of [209]

a better performance. In other words, a very deep model makes it possible to extract more complex and representative features at a lower cost and generalizes well to other domains. The proposed architecture is a homogeneous network including 16 or 19 layers that only leverages kernels of size 3×3 and 2×2 for convolution and pooling, respectively. The combination of two or three 3×3 convolutional layers has an effective receptive field of 5×5 and 7. This, in turn, simulates a larger filter that was used in previous architectures, while keeping the benefits of smaller filters. Moreover, as the spatial size of the input volumes at each layer decrease, the depth of the volumes increase from 64 channels up to 512 channels in high-level convolutional layers that reinforce the idea of shrinking spatial dimensions, but growing depth. One drawback of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters than previous architectures. One should note that the most of the network parameters belong to up-level fully connected layers. Further investigations showed that these FC layers can be replaced with 1×1 convolutional layers

without any performance downgrade. This modification is able to significantly reduce the number of the network parameters and its memory usage. Such a network was able to achieve the error rate of 7.3%. The global view of the VGGNet is illustrated in Fig 2-9.

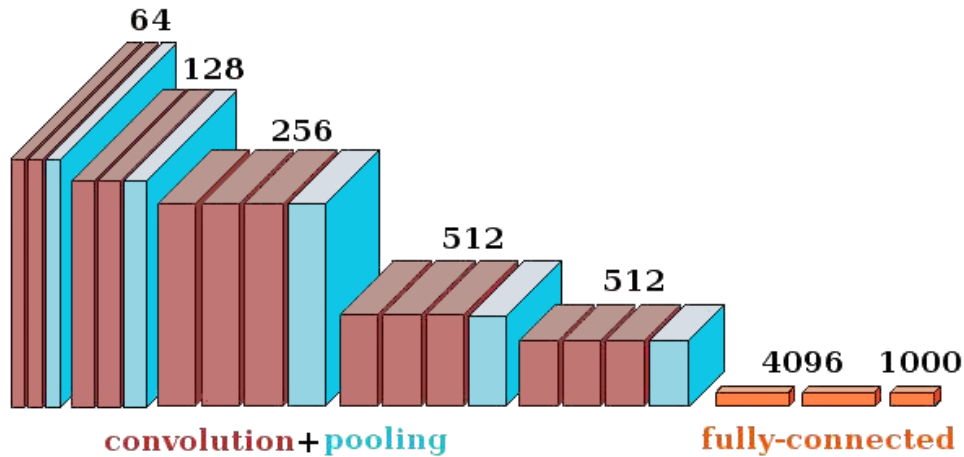


Figure 2-9: A repaint [28] of VGGNet [184] with 16 layers including five conv/pooling blocks and three FC layers

2.6.5 GoogLeNet

Although VGGNet gained a phenomenal accuracy on different kinds of visual recognition tasks, its deployment remains as a problem because of its huge computational requirements, both in terms of memory and time. The main contribution of the GoogLeNet [190] is the introduction of the *Inception Module* (Fig. 2-10) that dramatically reduced the number of parameters in the network. This concept is based on the idea that most of the activations (outputs) in previous deep architectures are either unnecessary or redundant because of correlations between them. There are some techniques to prune out such connections which would result in a sparse topology but kernels for sparse matrix multiplication are not optimized in common GPU-related packages that make their implementation even slower than their dense counterparts.

To resolve such a challenge, GoogLeNet utilizes inception modules to approximate a sparse CNN with a normal dense construction. This architecture starts with a sequential chain of convolution, pooling, and local response normalization operations, in a similar fashion to

previous models, such as AlexNet. Basically, at each layer of a traditional CNN, we need to make a choice of whether to have a pooling operation or a convolution operation. What an inception module provides is to perform all of these operations in parallel. Therefore, each inception module uses convolutions of different sizes 5×5 , 3×3 , and 1×1 to capture details at varied scales. The model contains nine of these modules, sequentially stacked, with two max-pooling layers along the way to reduce the spatial dimensions. But this strategy would end up with an extremely large depth channel for the output volume. The solution of the authors to address this issue is to embed 1×1 convolutional kernels known as the *bottleneck units* within the inception module to reduce the dimension of the input data, before feeding into bigger convolutional kernels.

Due to the depth of the architecture, the authors added two auxiliary classifiers branching from the main network structure. The purpose of these classifiers is to amplify the gradient signal back through the network, attempting to improve the earlier representations of the data. However, with the introduction of batch normalization, these classifiers have been ignored in updated versions. In addition, at the top of the inception modules, there is an average pooling mechanism instead of several FC layers to diminish the number of parameters. In this way, GoogLeNet has a factor of 12 times fewer parameters than AlexNet and was able to achieve the error rate of 6.67% on the 2014 ImageNet classification challenge.

2.6.6 ResNet and Residual Learning

As mentioned before, network depth is of crucial importance in neural network architectures. Unfortunately, deeper networks are more difficult to train. The issue arises from the way in which neural networks are trained through backpropagation. When a network is being trained, a gradient signal must be propagated backward through the network from the top layer all the way down to the bottom-most layer in order to ensure that the network updates itself appropriately. With a traditional network, this gradient becomes slightly diminished as it passes through each layer of the network. For a network with just a few layers, this is not an issue. In the case of a network with more than a couple dozen layers, however, the signal essentially disappears by the time it reaches the beginning of the net-

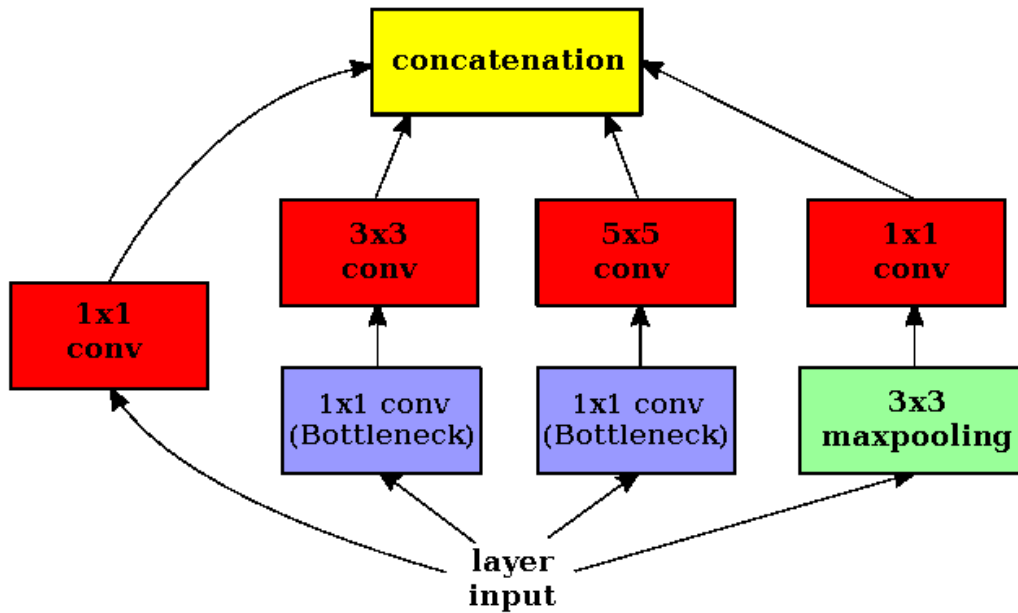


Figure 2-10: Internal structure of an inception module applied in GoogLeNet [190]. The figure is redrawn based on the figure 2 (a) of [190]

work again. So the problem is to design a network in which the gradient can more easily reach all the layers of a network which might be dozens or even hundreds of layers deep.

The *deep residual learning* framework is proposed to facilitate the learning process and make it feasible to design substantially deeper networks. The reasoning behind a residual block (Fig. 2-11) is that, while it is feasible to use a few stacked nonlinear layers to fit the desired underlying mapping $H(x)$, it is a good idea to let them approximate the counterpart residual mapping $F(x) = H(x) - x$. Consequently, it is possible to reformulate the underlying map as: $H(x) = F(x) + x$, which is easier to optimize.

Here, the term “ $+x$ ” can be implemented by a shortcut between the input and the output of the block that is called *skip connection* or *identity mapping*. This connection allows the gradient to pass backward directly. By stacking these layers, the gradient could theoretically skip over all the intermediate layers and reach the bottom without being diminished. Concerning the optimization process, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

Residual Network (ResNet) [81] is a really deep network with 152 layers that set new records in classification, detection, and localization through its wisely designed architec-

ture. Aside from the new record in terms of the number of layers, ResNet won ILSVRC 2015 with the amazing error rate of 3.6% which was for the first time beyond the human accuracy.

At the first layer, ResNet employs a 7×7 convolution with the stride of 2 to downsample the input by the factor of 2 similar to an ordinary pooling layer. Then the architecture is continued by three residual blocks before another downsampling step. The downsampling layer is also a convolution layer without the identity connection. This pattern is repeated several times to form the body of the ResNet. In the case of the ImageNet classification, the last layer is the average pooling which creates 1000 feature maps. The result would be 1000 dimensional vector which then fed directly into a Softmax classifier.

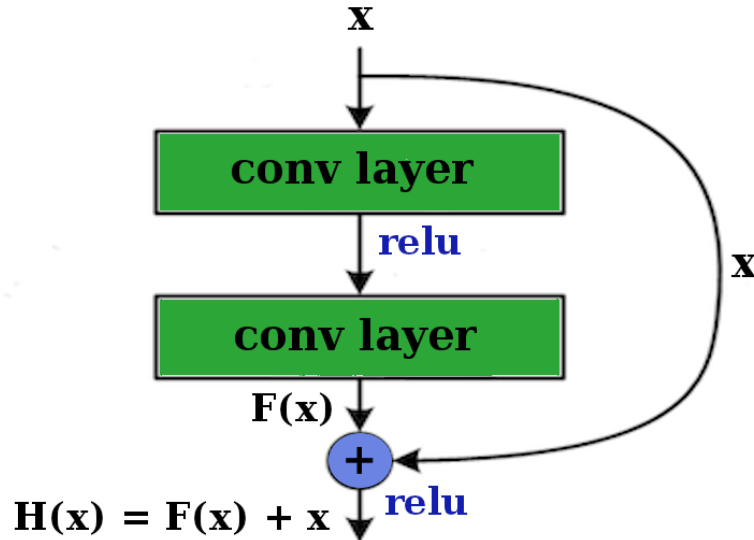


Figure 2-11: The structure of a residual block as applied in ResNet [81]. The figure is redrawn based on the figure 2 of [81]

2.6.7 DenseNet

As deep learning community understood connecting a skip connection from the previous layer improves the performance, DenseNet [88] architecture takes this insight to the extreme and proposed to connect the output of each layer to all the subsequent layers. In this way, there is always a direct route for the information backward through the network. This architecture makes intuitive sense in both the forward and backward settings. In the

forward-pass, a task may benefit from being able to get low-level feature maps in addition to high-level feature maps. In classifying objects, for example, a lower layer of the network may determine edges in an image, whereas a higher layer would determine larger-scale features such as the presence of faces. There may be cases where being able to use information about edges can help in determining the correct object in a complex scene. In the backward case, having all the layers connected allows us to quickly send gradients to their respective places in the network properly.

When implementing DenseNets, we cannot just connect everything though. Only layers with the same height and width can be stacked. So we instead densely stack a set of convolutional layers known as *Dense Block*, then apply a striding or pooling layer, and repeat this structure to form the whole network with a few dense block and a downsampling mechanism in between that is shown in Fig. 2-12. The network can perform well with dozens of layers where a traditional neural network fails.

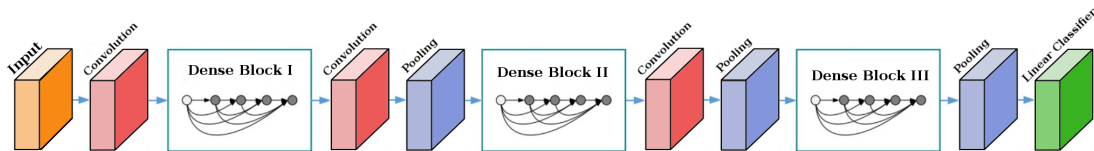


Figure 2-12: DenseNet [88] structure. The figure is redrawn based on the figure 2 of [88]

2.6.8 Fully Convolutional Networks (FCNs)

Fully convolutional networks (FCNs) [131] do not have any fully-connected layers, which are typically used at the end of CNNs for classification. FC layers require a fixed-size data representation provided by the previous layers to accomplish the intended task. Therefore, a network containing even one FC layer is restricted to the predetermined input dimensions. Instead, all the layers of an FCN are convolutional which let them accept any input size. So, their only restriction for the input size is the memory constraint.

The fundamental purpose of the FCN design is to perform *semantic segmentation*. In semantic segmentation or *pixel classification*, the output is in the same size as the input image, where each pixel is associated to one of the pre-defined class labels. At the end of an FCN,

there is often a *softmax* probability function that determines the most likely class of each pixel.

As can be seen in Fig, 2-13, after each block of convolutional layers, the height and width dimensions of the data get smaller and smaller, while the number of channels grows gradually. The main innovation in the FCN architecture is the application of the *deconvolution* or backward convolution mechanism to upsample each intermediate prediction of the network and make it prepared to be combined with the previous intermediate prediction. This combination pushes the finer spatial information of the lower intermediate predictions towards the end of the network so that the final network prediction exploits multi-scale spatial information to produce more accurate results and match the width and height of the original input image. Since deconvolution layers are just performing reversed convolutions, the more efficient upsampling process will be learned during backpropagation by parameter adjustment within those layers.

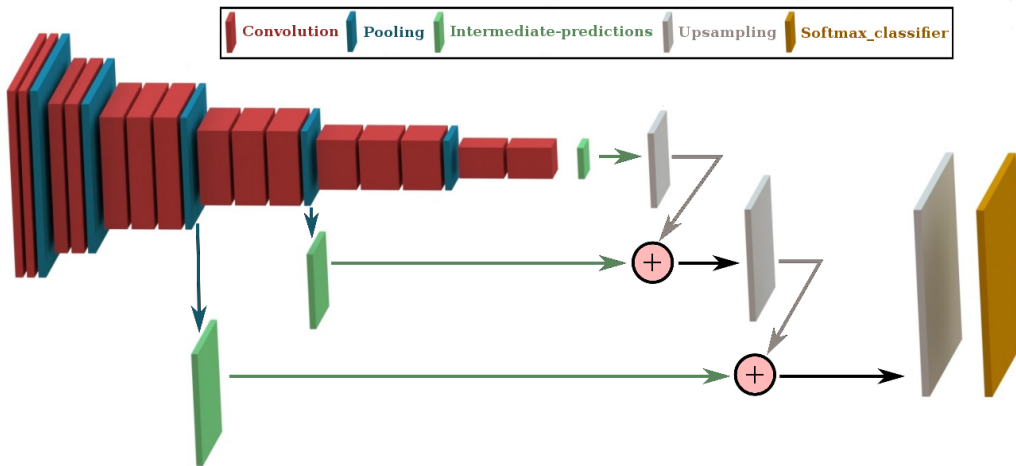


Figure 2-13: A repaint [28] of a Fully Convolutional Network (FCN) architecture [131] including a learnable upsampling mechanism

2.7 Representation Learning

Machine learning algorithms learn the solution of the problem from sample data. Hence, data representation has a great influence on their efficiency and has become a field in itself

in the machine learning community.

In general, a good representation is one in which the features within the representation correspond to the underlying causes of the observed data in a way that separate features or directions in feature space correspond to different causes. Under these circumstances, representation is able to disentangle the causes from one another, as in the case of training samples scarcity, an appropriate data representation is able to facilitate similarities identification among the data.

The other reason for the representation learning attractiveness is because of its ability to reveal the general priors of the data such as smoothness, sparsity, coherence, and dependencies [26]. For example, we are interested in representations whose entries are sparse or independent from each other since such a representation is easy to model. It worth to mention that although a representation that flawlessly separates the underlying causal factors of the observed data, may not possess the sparsity, in most of the AI tasks, these two characteristics occur simultaneously.

Many efforts of the computer vision community are concentrated on the design of the pre-processing pipelines and data transformations that lead to a proper data representation. This process that is also known as *manual feature engineering* can be considered as the construction of the *local image descriptors* [63] such as Scale-Invariant Feature Transform (SIFT) [132], Speeded Up Robust Features (SURF) [22], Local Binary Pattern (LBP) [8] and Histogram of Oriented Gradients (HOG) [52].

Image descriptors are widely used to detect the local image properties while maintaining robustness to various kinds of transformations including scale changes, viewpoint changes, image blur, in-plane rotation, noise, and illumination. In this context, visual descriptors are normally used in three different modes. A *sparse descriptor* detects salient interest points in a given image and then samples a local patch and describes its invariant features. SIFT is the most commonly used sparse descriptor. The second approach is based on computing on a dense grid of uniformly spaced cells. HOG and SIFT are widely used for this task. Finally, texture descriptors are obtained by regular sampling of the input image or region. In recent years, LBP is widely used as the dense texture descriptor, but can also be used as a sparse local descriptor similar to SIFT or computed on a grid-like HOG.

In spite of its difficulty, manual feature engineering played a key role in data preparation by taking advantage of human ingenuity and prior knowledge. This procedure involves a delicate blend of domain knowledge, intuition, and mathematical abilities. So, most of the time it is highly expensive, time-consuming, and error-prone.

In the case of a feed-forward network, the main responsibility of the layers is to provide a suitable representation of the data to the final layer which is normally a classifier. The outcome of a successful learning process is the gradual formation of this representation inside of the network's structure. Consequently, in a complex classification task, where classes are not linearly separable in the input space may become linearly separable in the last hidden layer.

In some learning strategies such as supervised learning of the neural networks, learning process does not involve any explicit condition on the intermediate features. In contrast, some kinds of learning approaches are intentionally designed to form a specific data representation. In all the cases, it is important to consider a trade-off between preserving as much information about the input as possible and providing intended representation properties.

2.8 Feature Encoding

While *feature extraction* is the process of representing input data in a reduced form with minimum possible redundancy to facilitate the solution of the pattern detection, classification and recognition problems, *Feature encoding* is the compression process of the resulting feature vectors in order to reduce the computational complexity and, more importantly, to avoid overfitting risk. This compression mechanism is an important strategy to achieve high performance in image processing operations.

In CNN architecture, raw image pixels are first sent through convolution layers that operate as different local feature extractors. So far, the output feature maps preserve a relative spatial arrangement of the input information. The resulting globally ordered features are then concatenated and fed into the final decision maker such as a classifier. At this point, the representation still preserves a great deal of global spatial information. Though max-pooling within each feature map helps to improve invariance to small-scale deformations,

invariance to larger-scale, more global deformations might be undermined by the preserved spatial information and the final CNN representation is still fairly sensitive to global translation, rotation, and scaling. Even if one does not care about this lack of invariance for its own sake, it directly translates into a loss of accuracy for classification tasks. Nevertheless, this framework has achieved great success in image classification, object recognition, scene understanding and many other applications, but is typically not ideal for recognizing dynamic patterns such as textures and materials. This is basically due to the need for a spatially invariant representation describing the feature distributions instead of concatenation. Therefore, an *orderless* feature encoding layer is desirable for such operations.

In the generic visual categorization, a Bag-of-Words (BoWs) [51] corresponds to a histogram of the number of occurrences of particular image patterns in a given image. This method is motivated by an analogy to learning methods using the BoWs representation for text categorization [49] and benefits from clustering to obtain quite high-dimensional feature vectors for a classifier. Ideally, these feature vectors are designed to maximize classification accuracy while minimizing computational effort.

Intuitively, BoWs and CNNs lie towards opposite ends of the *orderless* to *globally ordered* spectrum of visual representations. With the recent improvements of the deep learning framework, hand-engineered features and filter banks are replaced by pre-trained CNNs, and BoWs are replaced by the robust residual encoders such as Vector of Locally Aggregated Descriptors (VLAD) [92]. VLAD is a representation that encodes by the residual vectors with respect to a dictionary, and Fisher Vector [158] can be formulated as a probabilistic version of VLAD. Both of them are powerful feature encoding methods for image retrieval and classification.

2.9 Transfer Learning and Fine-tuning

The training process of a deep neural architecture may take too much time due to the huge number of parameters. In addition, a supervised learning task requires a sufficient number of training samples which is not available in every domain. So, it is common to train a big architecture on a huge existing data set of the source domain and transmit the gained

knowledge to a target domain in which the generalization improvement is the main concern. In the normal transfer learning, the input is the same but the targets are different. In this situation, one possible solution is to remove the last domain-specific layers and then treat the rest of the network as a fixed hierarchical feature extractor for the target data. The second strategy is to not only cut off the classifier on top of the network but also fine-tune the weights of the pre-trained network by further training. It is feasible to fine-tune all the layers of the network or to keep some of the earlier layers fixed and proceed to learn the higher-level section of the network. Sometimes, however, what is shared among the various domains is not input but output semantic. In such a case, it is more reasonable to share the parameters of the upper layers and perform a domain-specific preprocessing on the target task.

As mentioned before, initialization of a big network is a challenging task that may influence the subsequent training accuracy. Here, a pre-trained model can also be used for a proper network initialization in a similar learning problem space.

2.10 Region Based CNNs

The main weakness of the visual classification is that it is designed to detect only one class per image. In contrast, object detection aims to find all the existing samples of semantic objects in the scene. One easy way to perform object detection is the application of the regression techniques to determine the location of each instance with four numbers, (h, w, x, y) , representing height, width and the coordinates of the reference point (normally upper-left corner) of its bounding box. Unfortunately, the unknown number of the existing objects varies the output size of the regression solution and makes it difficult to use regression for the detection task.

The more plausible solution is to select *random* areas in the scene and feed them separately into a classifier to check if each region contains one of the pre-defined object classes or not. Before deep learning solutions, the most successful approaches utilized the popular *sliding window* model [66], in which a computationally efficient classifier examines object presence in every determined image window. Sliding window classifiers scale linearly with the

number of windows examined, and while single-scale detection requires classifying around $10^4 - 10^5$ windows per image, the number of windows grows by an order of magnitude for multi-scale detection.

The proposed solution to make a trade-off between computational expense and the recognition accuracy is the application of *region proposal* approaches. Intuitively, object instances share common visual properties that distinguish them from the background. So, it makes sense to design or train a model which produces a set of proposal regions that are likely to contain objects. If high accuracy can be reached with a much lower number of windows than used by sliding window mechanism, that will speed-up the recognition process and makes it possible to use more sophisticated classifiers.

Two well-known group of techniques that lead to efficient generation of region proposals are *window scoring proposal methods* [12] and *grouping proposal approaches* [17]. In the first step of the window scoring mechanism, a huge number of bounding boxes are generated. Then, some low-level object features will be used to calculate objectness score in those boxes. In the next step, all the bounding boxes will be sorted based on their scores to select the most likely locations of the object instances.

In contrast, grouping proposal methods decompose the input image into many fragments, and then utilize low-level similarity features to merge these fragments. For instance, *Selective Search* (SS) [195] determines the original regions through graph-based image segmentation [67], and combines them using hierarchical grouping algorithm. In the last step, the selective search is used to estimate the possible position of the object. This approach has been widely used in several object detectors, including *R-CNN* [70] and *Fast R-CNN* [69] approaches.

2.10.1 R-CNN

Region-based Convolutional Neural Network (R-CNN) [70] is a highly successful and widely used method that couples object proposals with CNNs. This architecture splits recognition process into separated components, the region proposal stage and the classification phase.

As illustrated in Fig. 2-14, in the region proposal step, the selective search method is used to generate around 2000 different regions that have the highest probability of containing an object. In the next stage, each proposal is warped into a fixed-size frame that can be fed into a dedicated CNN to extract a feature vector for the corresponding region. This vector which is normally the output of the last FC layer in the CNN is then used as the input to a set of linear SVMs that are trained for each class to produce final classification results. Since the original bounding box proposals may need further refinement, the vector also gets fed into a bounding box regressor to obtain the most accurate coordinates.

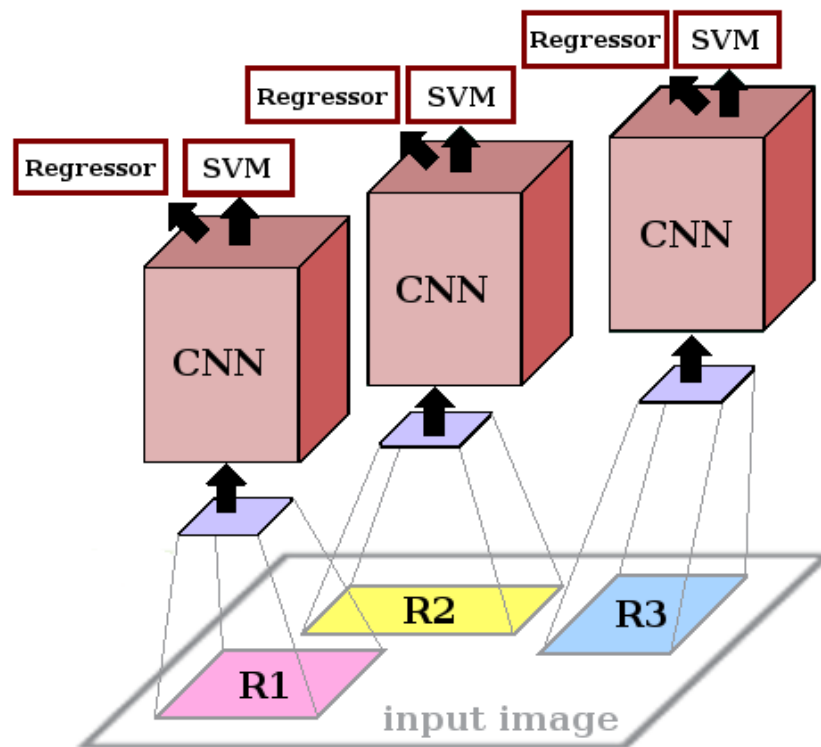


Figure 2-14: A repaint [28] of R-CNN mechanism [70] for object recognition

Unfortunately, R-CNN has a multi-stage training pipeline that is computationally expensive and time-consuming. During the training process, we need to fine-tune the CNN on each of the object proposals, fit the SVM classifiers to feature vectors and learn the bounding-box regressors. Hence, training phase requires several days and hundreds of gigabytes, while the CNN architecture used here (AlexNet) is not even as big as recent models.

Moreover, at the test-time, features should be extracted from each object proposal in each

test image. This setting is slow because it performs a separate forward-pass for each object proposal, without sharing computation.

2.10.2 Fast R-CNN

Fast R-CNN [69] proposed a new training algorithm that fixes the drawbacks of R-CNN and improves the training speed and accuracy. In this architecture, instead of feeding each region proposal into a separate CNN, one network receives the whole input image once and generates one global feature map at its last convolutional layer. Afterwards, for each region proposed by selective search, a *region of interest* (RoI) pooling layer extracts a fixed-length feature vector from the global feature map. The RoI pooling layer employs max pooling to convert the features in a region of interest into a small feature map of size $H \times W$ where both H and W are tunable hyper-parameters.

In the last step, each feature vector is fed into a sequence of FC layers that finally branch into two sibling output layers. The first branch is a softmax probability estimator on all object classes and the background, while the second branch generates 4 real-valued numbers for each of the object classes. Each set of 4 values encodes refined bounding-box positions for each possible class.

As can be seen in Fig. 2-15, an important superiority of the Fast R-CNN pipeline over R-CNN structure is that all the network parameters can be trained together using the log loss function of the classification and $L1$ loss function of the regression.

2.10.3 Faster R-CNN

Both R-CNN and Fast R-CNN require an external region proposal method that is found to be the bottleneck of the overall recognition process. In contrast, Faster R-CNN [163] proposed a Region Proposal Network (RPN) that takes the output feature maps of the detection networks and feed them into some extra convolutional layers to generates region proposals. In this architecture, the input image goes through a CNN which leads to a set of feature maps on the last convolutional layer. Then, through some extra convolutional layers, a sliding window is run spatially on these feature maps. The size of sliding window is $n \times n$. For

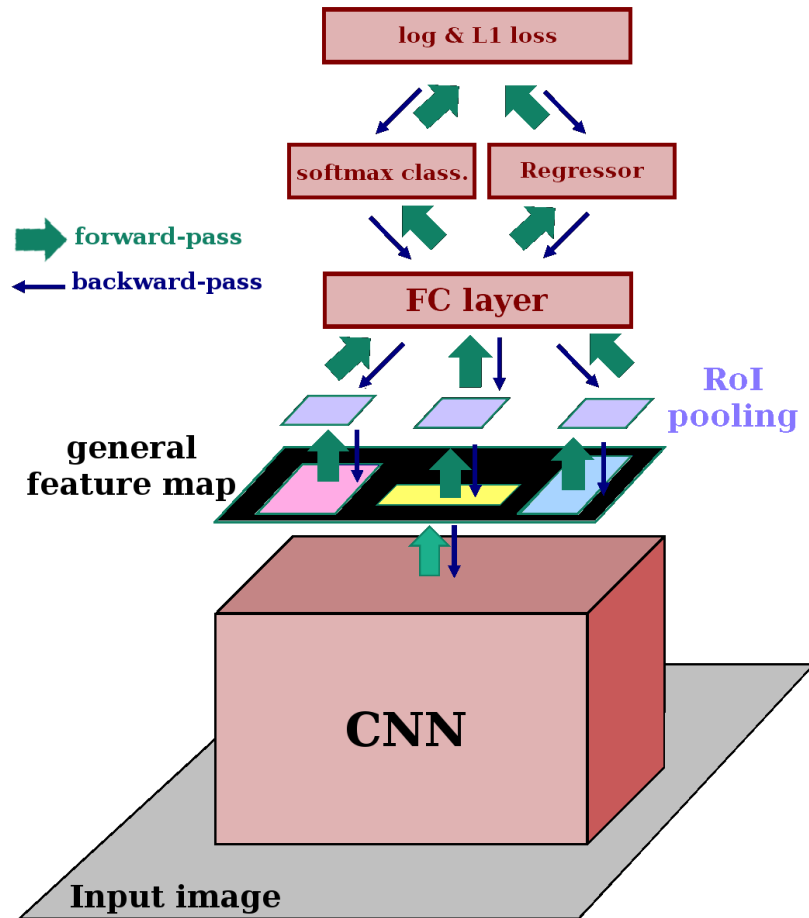


Figure 2-15: A repaint [28] of Fast R-CNN framework [69]

each sliding window, a set of 9 anchors will be generated which all have the same center (x, y) but different aspect ratios (AR1, AR2, AR3) and scales (S1, S2, S3) as shown in Fig. 2-16. The multi-scale anchors' technique is the key component for sharing features without the extra cost of addressing scales. The output of the RPN is a bunch of boxes that will be examined by a classifier and regressor to eventually check the occurrence of the objects. Since the RPN shares the most computation with the object detection network, the time of generating region proposals in RPN is much smaller than selective search. These proposals are then fed into the RoI pooling of Fast R-CNN mechanism for further processing. The RPN network is initialized with an ImageNet pre-trained model and will be fine-tuned via a joint training process that alternates between region proposal task and the object detection. Since this architecture is one of the basic components of our hybrid solution for interactive

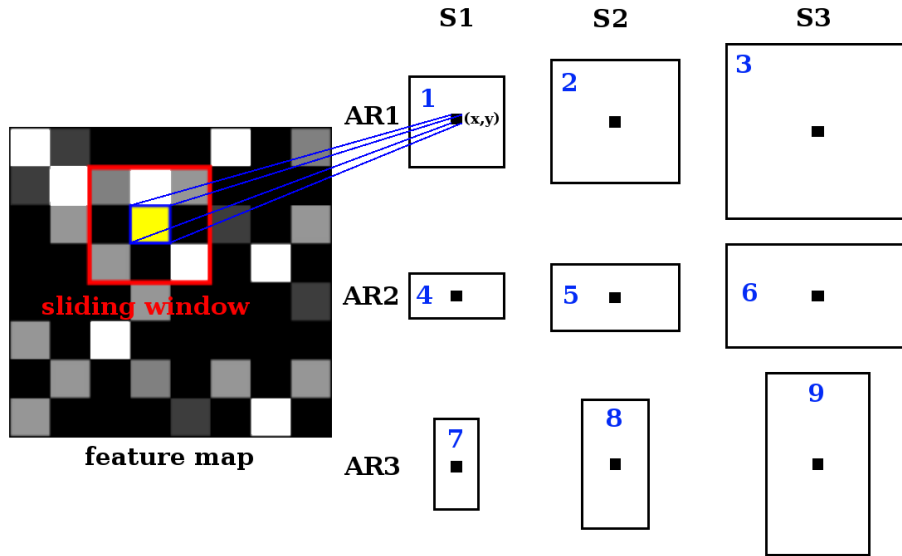


Figure 2-16: A repaint [28] of the anchor generation on CNN feature maps in Faster R-CNN approach [163]

region segmentation and captioning, we discuss its internal dynamics in more depth later in context with our method.

2.11 Sequence Modeling

In a traditional neural network, we assume that all inputs and outputs are independent of each other. But the vast majority of interactions in our daily lives can be found in the form of sequential data. In other words, we often want to turn an input sequence into an output sequence that lives in a different domain. For example, we might want to take a sequence of sound pressures and turn it into a sequence of word identities as we do in speech recognition. In some cases, there is no target sequence. Instead, the output sequence is simply the input sequence with an advance of one time step to predict the next term in the input signal. One should note that in temporal sequences, there is a natural order to do such predictions while in static data such as images, it is not clear how can we predict pixel intensities based on the rest of the image. This approach is somewhere between supervised and unsupervised learning. In that sense, we used supervised learning methods to predict the next element of the sequence, while we do not need labeled data to do so. In other

words, the label is not in the data itself, but in the context of the data. So, for a proper transformation, one should not only convert inputs into outputs but also retain relations and dependencies. In the feed-forward architectures, the model is able to understand sequential relations only when the sequence is smaller than or equal to the capacity of the input layer. That is, feed-forward networks cannot memorize correlations beyond a time step. But, there are some useful strategies to process sequences using memory-less methods and hidden state models.

2.11.1 Autoregressive Models

Memory-less autoregressive models [10, 56] are proposed to predict next element of the sequence (y_t) as the weighted average of a fixed number of previous elements without using neural network architecture. For example :

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \epsilon_t \quad (2.13)$$

These previous elements can be organized in the shape of some individual values or a vector. In this regression model, the response variables in the previous time steps have become the predictor and the errors have our usual assumptions about errors in a simple linear regression model. The order of an autoregressive model is the number of immediately preceding elements in the sequence that are used to predict the element of the present time. Feed-forward networks are able to generalize autoregressive models using one or more layers of non-linear hidden units and perform more complicated predictions [99].

2.11.2 Hidden State Models

Memory-less models are not the only models that can be used to predict sequences. Another approach for sequence prediction is to develop models that are able to generate sequences. Normally, this kind of models exploits some hidden state that evolves according to its internal dynamics to produce a proper sequence of elements. The hidden state is a place to memorize the information for a long time, so there is no bound to keep track of long-term

dependencies.

Since the internal dynamic of the system has some noises, it is almost impossible to infer the hidden state of the system based on output observations. So the solution is to infer a probability distribution over the space of all possible hidden state vectors. This leads to very difficult computations. There are only two types of the hidden state models for which these computations are tractable, *Linear Dynamical Systems* [98] and *Hidden Markov Models* [161]. In those models, we assume that the data is generated by the model. So, we infer what the hidden state of the model must be, in order to generate that data.

Linear Dynamical Systems

A Linear Dynamical System (LDS) [98] is a generative model that has a real-valued state vector with internal linear dynamic (a.k.a. Evaluation function) including Gaussian noise and generates its outcome based on that while the hidden state evolves probabilistically. Since the linear transformation of a Gaussian distribution leads to another Gaussian distribution and linear dynamic systems include only Gaussian noises, their distribution over the hidden state is a full covariance Gaussian that is hard to compute. Fortunately, there is an efficient recursive technique for the closed-form solution of this Gaussian covariance called *Kalman filtering* [207].

Hidden Markov Models

A Hidden Markov Model (HMM) [161] has several states and the system is always in exactly one of those states that form a *one-of-N* choice. The transition between possible states is controlled by a transition matrix that includes a bunch of probabilities. Consequently, the output of the model is completely stochastic meaning that the current state cannot exactly determine what output it produces. So, hidden states are covered by the probabilistic nature of the system. Fortunately, it is an easy task to represent the probability distribution across N states with N numbers.

For the prediction of the system output, there is an easy method based on dynamic programming called *forward algorithm* that is able to use system observations to compute the probability distribution across the hidden states. The main limitation of the HMMs is the

limited number of their hidden states. At each time step, the model should select one of the hidden states. So, with N hidden states, the maximum amount of the system memory is $\log(N)$ bits that confine its power to maintain enough information that is needed to detect long-term dependencies.

2.11.3 Recurrent Neural Networks

The feed-forward title points to this convention that there is no cyclic data flow through the network. Consequently, it is not possible in this kind of the networks to feedback the model output into itself. Cyclic paths are essential for the task of sequence analysis that is provided in *Recurrent neural networks* (RNNs) to help them memorize what has been calculated so far and recognize long-term dependencies.

RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. As depicted in Fig. 2-17, a useful way to visualize an RNN is to consider the updated graph formed by *unfolding* the network along the input sequence. By unrolling we simply mean that we write out the network for the complete sequence. Here, x_t is the input, y_t is the output, s_t is the hidden state, and w_t is the set of network parameters at time step t . Because the parameters are shared by all the time steps, the gradient at each output depends not only on the calculations of the current time step but also the previous time steps. So, in order to calculate the gradient for every time step, it is necessary to backpropagate through all the previous time steps and sum up the gradients. This version of backpropagation is known as *Backpropagation Through Time* (BPTT).

Bidirectional RNNs [178] come from this idea that the output at time t may not only depend on the previous elements in the sequence but also future elements. For instance, to predict a missing word in a sequence we may need to look at both the left and the right context.

Comparing to previously mentioned hidden states models, RNNs have a much more efficient way of remembering information due to the combination of two properties. First, they are easily able to memorize several topics at once by using distributed hidden states in which lots of units can operate concurrently, whereas, in HMMs, we had just one active

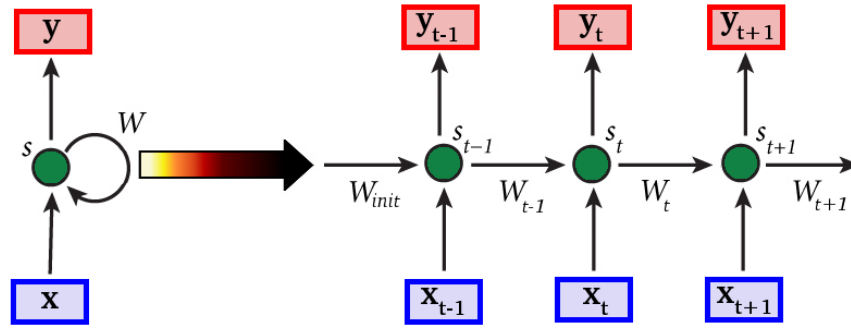


Figure 2-17: Unrolling of RNN [28]

unit. The second superiority of the RNNs is the nonlinear functionality that helps their hidden state to have much more complicated dynamic allows them to simulate all the possible situations.

It is worthwhile noting that in previous stochastic generative models, both the dynamic of the hidden state and the production of system observations involve intrinsic noise. In addition, despite their stochastic action, their posterior probability distribution over hidden states is a deterministic function of the data that the system has seen so far. In other words, the inference algorithm of these systems ends up with a probability distribution which is a bunch of numbers that forms a deterministic function of the data. In RNNs, we also get a bunch of numbers that are the deterministic function of the data and we can think of these numbers as the hidden state of the RNN.

There are some specific behaviors that we can expect from RNNs. For example, RNNs are able to oscillate [193]. That means, for a well-defined set of parameters, they can adapt their internal parameters to learn and then replicate autonomously certain external periodic signals. So, RNNs can be very useful tools for periodic tasks such as motor control in robotics.

Dynamic behavior of continuous attractors is another interesting property of RNNs. Continuous attractors form a set of connected equilibrium points of a network. Continuous stimuli, such as orientation, moving direction, and the spatial location of objects could be encoded as continuous attractors in RNNs [211]. Moreover, RNNs have the potential of behaving chaotically that can be exploited to generate randomness in adversarial environ-

ments. More importantly, RNNs can learn to execute a great number of small programs by exploiting different parts of their hidden state. Each of these mini-programs can be used to capture a small piece of knowledge by parallel implementation and interaction.

The most common application of the sequence analysis is the field of Natural Language Processing (NLP) in which every sentence can be modeled as a sequence of words.

2.11.4 LSTM

The dynamic state of a neural network is a short-term memory and we want to make it lasts for a long time such as hundreds of time steps. Long-Short Term Memory (LSTM) mechanism [84] contains special modules that are designed to allow information to be gated in and out when needed while in the intermediate intervals the gate is closed so that information that arrives does not interfere with the remembered state.

Each LSTM memory unit (Fig 2-18) is a gated environment that holds information outside the normal flow of the recurrent network. These memory units have many variations, but we will stick to a simple one that consists of three gates: *input* (i_t), *forget* (f_t), and *output* (o_t). It also includes an internal cell state (c_t) which plays a similar role for the memory cell that h_t plays for the network. Those gates act on the signals they receive and block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning the process. That is, memory units learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.

Information gets into the cell whenever the input gate is on. The rest of the recurrent network including network parameters (w_t), current input (x_t) and the network previous hidden state (h_{t-1}) determine the state of the input gate. In other words, when the network wants to store a piece of information, it turns on the input gate. The information stays in the memory cell as long as the forget gate is off. Also here, the rest of the system determines the state of the forget gate. The information can be retrieved from the memory cell when its output gate is on which again is a logistic unit controlled by the rest of the recurrent

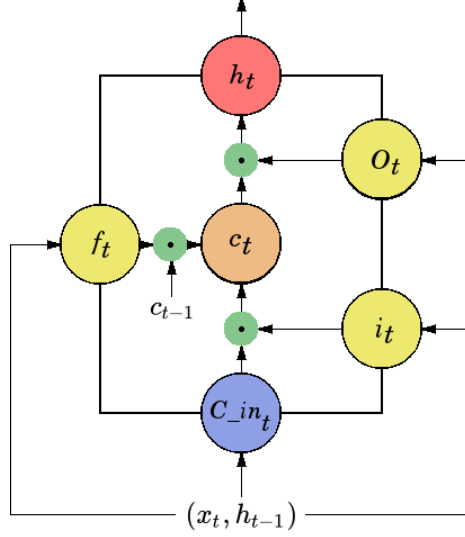


Figure 2-18: A repaint [28] of LSTM gating mechanism [84]

network. Actually, the memory cell stores an analog value and keeps writing that value to itself at each time step while the forget gate is off in the form of a coefficient with a value near to one. When the system decides to clean the stored information, all it needs is to change the value of that coefficient to zero. The internal dynamics of a memory cell can be formulated by the following set of equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (2.14)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (2.15)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (2.16)$$

$$C_in_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_{c_in}) \quad (2.17)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot C_in_t \quad (2.18)$$

$$h_t = o_t \cdot \tanh(c_t) \quad (2.19)$$

The multiplicative structure of the LSTM units especially in terms of the nature of the forget gate, make them convenient to backpropagate through and help the information to travel back through hundreds of time steps. As the result, the system will be able to learn the

proper use of the LSTM units over many time step to memorize long-term dependencies.

2.12 Deep Learning Frameworks

The art and science of training neural networks from large data sets in order to make predictions or classifications have experienced a major transition over the past several years. One interesting point related to deep learning frameworks is that the first generation was mostly built in academia for research purposes, while the next generation is originated from the IT industry and have more powerful properties. Some current widely used deep learning frameworks will be examined here and compared, across various features, such as the native language of the framework, multi-GPU support, and aspects of usability.

2.12.1 Caffe

Built with expression, speed and modularity in mind, *Caffe* [94] is one of the first deep learning libraries developed mainly by Berkeley Vision and Learning Center (BVLC). It is a C++ library which also has a pretty useful Python interface and finds its primary application in modeling CNNs and R-CNNs. Unfortunately, this framework is not intended for other deep learning applications such as text, sound or time series data. Nevertheless, the major benefit of this library is that the user can benefit from a large repository of pre-trained neural network models suited for a variety of image classification tasks and available for immediate use known as *Caffe Model Zoo*. Moreover, models and optimization are defined by configuration without hard-coding.

Following the footsteps of Caffe, Facebook also recently open-sourced *Caffe2* [1], a new light-weight, modular deep learning framework which offers greater flexibility for building high-performance deep learning models.

2.12.2 Theano

As a primary deep learning library, *Theano* [11] is a low-level Python-based framework that is particularly good when it comes to numerical computation. Unlike all other deep

learning frameworks, Theano computes the gradient when determining the backpropagation error by deriving an analytical expression. This eliminates the accumulation of error during successive derivative calculations using the chain rule, which other frameworks accrue due to their use of numerical methods. Although the framework supports the use of multiple GPUs, configuring Theano to use more than one GPU requires a cumbersome workload. There are several open-source deep libraries that have been built on top of Theano, including *Keras* [42], and *Lasagne* [174]. These libraries attempt to provide an easier to use API on top of Theano's non-intuitive interface. Recently, it is announced that development on Theano is ceased.

2.12.3 Tensor Flow

While new to the open source landscape, Google's *TensorFlow* [6] deep learning framework has been in development for years as proprietary software. It was developed originally by the Google Brain Team for conducting research in machine learning and deep neural networks. Similar to the most deep-learning frameworks, TensorFlow is written with a Python API over a C/C++ engine that makes it run faster. This framework offers a good amount of documentation for installation, as well as learning materials which are aimed at helping beginners understand some of the theoretical aspects of neural networks, and getting TensorFlow set up and running relatively simple.

Unlike any other framework, TensorFlow has the ability to do partial sub-graph computation, which involves taking a sub-sample of the total neural network and then training it, apart from the rest of the network. This is also called *Model Parallelization* and allows for distributed training on a cluster. It is worth noting that Keras is also able to support TensorFlow.

2.12.4 Torch

Torch [46] is originally developed at NYU, and is based upon the scripting language *Lua*, which was designed to be portable, fast, extensible, and easy to use in development. Lua was also designed to have an easy-to-use syntax, which is reflected by Torch's syntactic

simplicity. This framework features a large number of community-contributed packages, giving it a versatile range of support and functionality. As an advantage, Torch is able to import trained neural network models from Caffe's Model Zoo. This package also provides better debugging tools.

Torch is better for debugging than Theano and TensorFlow. This is because Torch is built for *automatic differentiation*, while TensorFlow and Theano use *symbolic computation*. The latter means that when you code an operation symbolically, it is not actually computed when that line of code is interpreted. It first needs to be compiled, optimized, and translated to C/CUDA as a computational flow graph and then it can be executed. This is what makes debugging painful as an error in the graph is harder to associate to a line in the code. The compilation process not only adds a layer of complexity to the code but also requires time, which is itself a debugging bottleneck. Both Theano and TensorFlow make huge efforts to keep this compile time as trivial as possible, while Torch has no compile time and is not a compiler.

Recently, the Python interface of Torch, called *PyTorch* [157], has found popularity and is gaining rapid adoption. PyTorch is a python package that provides two high-level features: *Tensor computation* and Deep Neural Networks built on a tape-based *autograd system*. For tensor computation, PyTorch provides a tensor structure that can live either on the CPU or the GPU, and accelerate compute by a huge amount. This data structure provides a wide variety of routines to accelerate and fit scientific computation needs such as slicing, indexing, math operations, linear algebra, and reductions. PyTorch is not a Python binding into a monolithic C++ framework. It is built to be deeply integrated into Python That means the PyTorch user is able to write new neural network layers in Python itself using python packages such as numpy, scipy, scikit-learn. The memory usage in PyTorch is much more efficient compared to other frameworks. It utilizes custom memory allocators for the GPU to make sure that deep learning models are maximally memory efficient. This enables the user to train bigger deep learning models than before.

2.12.5 DL4J

Deep Learning for Java (DL4J) [191] is a popular deep learning framework developed in the widely used programming language, Java, and supports other JVM languages as well. This framework is widely used as a commercial, industry-focused distributed deep learning platform. DL4J can bring together the power of the whole Java ecosystem to perform efficient deep learning, as it can be implemented on top of the popular Big Data tools such as *Apache Hadoop* and *Apache Spark*. It also has a dedicated open-source numerical computing library called N-Dimensional Arrays for Java (ND4J) that is claimed to be more efficient than the prominent Python-based Numpy library. While both Torch and DL4J employ parallelism, DL4J's parallelism mechanism is able to automate the setting up of worker nodes and connections, allowing users to bypass libraries while creating a massively parallel network on Spark, or Hadoop. In the DL4J environment, shallow neural nets such as restricted Boltzmann machines, convolutional nets, autoencoders, and RNNs can be added to one another to create deep nets of varying types. It also has extensive visualization tools and a computation graph. DL4J can import models from Tensorflow and other Python frameworks if they have been created with Keras.

Chapter 3

Deep Interactive Region Segmentation and Captioning

In this chapter, we provide a deep hybrid interactive architecture that is able to not only generate a wide range of linguistic descriptions for different regions of the input image, but also detect user priority known as *User Intended Region* (UIR), segment it with the high accuracy and provide the most accurate caption for its visual content. This work is appeared in proceeding of the thirteenth international conference on Signal Image Technology and Internet-based Systems (SITIS 2017) [30] in Jaipur, India. The final publication is also indexed in IEEE Xplore.

3.1 Introduction

The human visual system including eyes, optic nerves and brain is able to easily detect, separate and describe each object of a scene. Furthermore, the human observer is easily able to provide detailed explanation about different parts of an image which is a hard task in artificial intelligence. Inspired by these natural abilities, *interactive region segmentation and captioning* is the task of parallel detection, localization and description of the user visual interests. This procedure can be exploited in several applications such as automatic image annotation and retrieval [187] and providing a better understanding of the virtual world for visually impaired people [206, 200].

In this chapter, we propose our novel hybrid deep architecture for integrated detection, segmentation and captioning of the user preferences where the amount of the user interactions is limited to one or a few clicks. As depicted in Fig. 3-1, our hybrid approach of interactive region segmentation and captioning introduces a new class of models where the user visual concentration can be specified and described, simultaneously.

With the increasing popularity of deep learning architectures [107, 190, 81, 70, 69, 163, 131, 88, 93, 203], both automatic detection and captioning objectives have attracted a new wave of considerations [95, 41, 200, 202]. Convolutional Neural Networks (CNNs) [115] have presented the ability to construct numerous visual features in different levels of abstraction through supervised learning. Such a property introduces the CNN-based models as the fast and scalable automatic feature generators that are able to reach near-human performance in various computer vision tasks.

In contrast to CNNs, Fully Convolutional Networks (FCNs) [131, 93], are able to maintain spatial information which is crucial to perform a pixel-level prediction such as semantic segmentation [127, 120], object localization [179], depth estimation [59] and interactive segmentation [203]. Furthermore, Recurrent Neural Networks (RNNs) [84] are excellent tools to memorize long term dependencies which is essential for exploring the continuous space of natural languages.

Recently, CNN-RNN models [100, 95] have been proposed to wrap detection and captioning tasks in an end-to-end learnable platform. However, up to now the results appear to be mostly an unorganized and overcrowded set of captions and bounding boxes that are not easily understandable, especially in the presence of several overlapping region proposals see e.g. Fig. 3-1 (b). In addition, they do not involve user intentions. In other words, the user is not able to involve personal preferences to exclude areas that are out of interest. It includes both segmentation and captioning operations.

3.2 Deep Region Detection

The first step in our hybrid solution for interactive region segmentation and captioning is the full understanding of the visual contents of the scene. To do so, we need to detect and

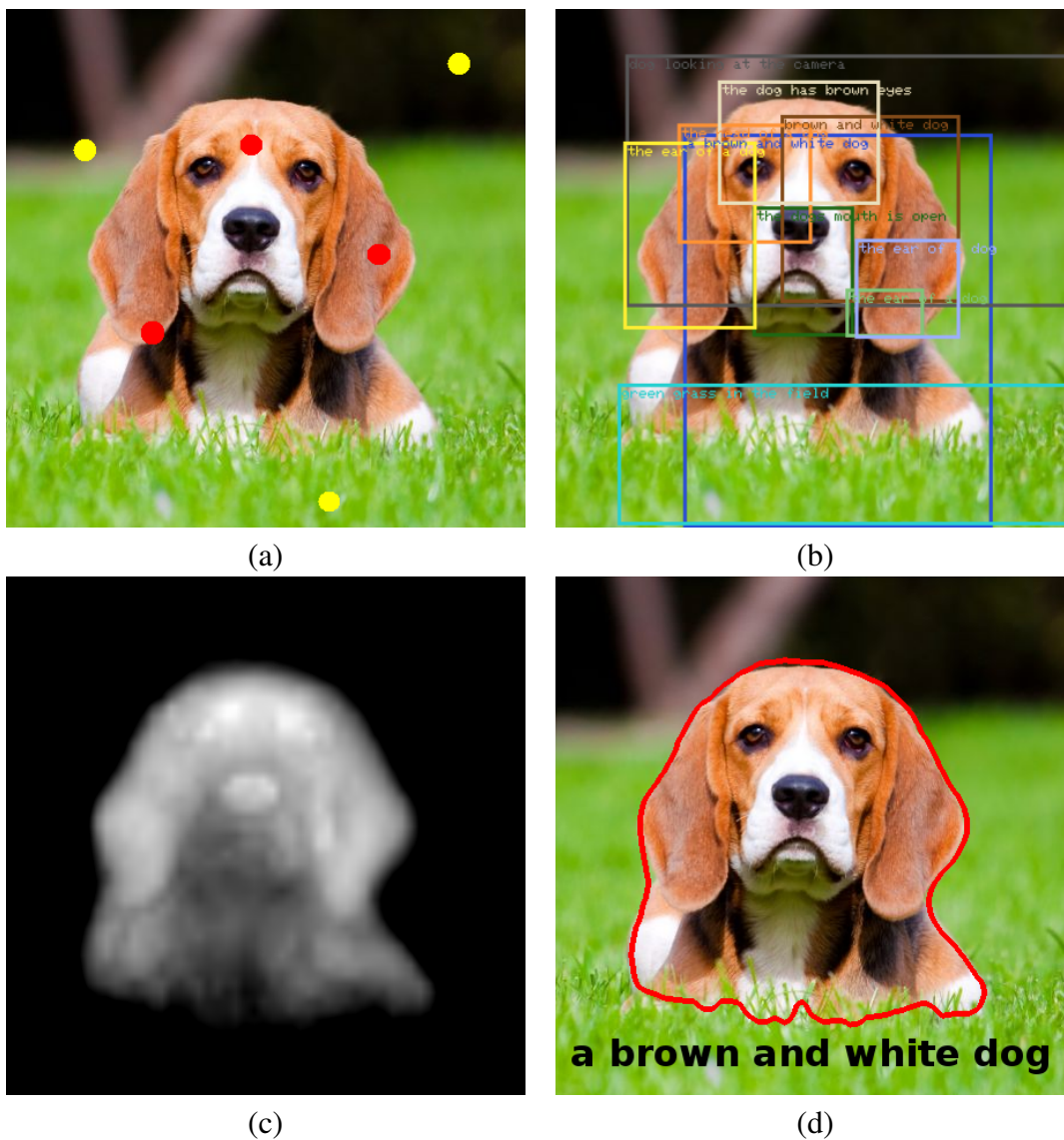


Figure 3-1: (a) Input image [3] including positive and negative user clicks, (b) output of the dense captioning process [95], (c) probability map of our deep interactive segmentation framework considering user priorities, and (d) the final output of our hybrid model including highlighted user intended region (UIR) and its proper description.

locate all the important objects and regions in the input image.

Object instances share common visual properties that distinguish them from the background. So, it makes sense to design and train a model which produces a set of *region proposals* that are likely to contain objects. A related point to consider is that the concentration on the most salient objects in the input image reduces the computational complexity of the subsequent task of recognition and makes it possible to use more sophisticated and powerful machine learning algorithms.

3.2.1 Generic Input Representation

In almost all the machine learning algorithms, the representation method of the data has a huge influence on the output quality. Consequently, many efforts of the machine learning community are concentrated on various data transformation techniques that provide a data representation for effective application of machine learning approaches.

As mentioned before, in deep learning framework, the fundamental structure to generate such an input data representation is the Convolutional Neural Network. In our hybrid architecture, a VGGNet [184] consists of 13 convolutional layers encodes the visual contents of the image in the form of many convolutional features that are extracted in different abstraction levels. In such a situation, the quality of the extracted convolutional features is heavily depends on the richness of the training data. So, normally such a network is initialized by the weights of a similar network that is trained on a big dataset such as ImageNet [172] for the task of object classification. To clarify which kind of knowledge will be transformed by the application of such a pretrained CNN, one should note that the data that is used to train models for the ImageNet classification task is consist of 1.2 million photographs, collected from flickr and other search engines, which is hand labeled with the presence or absence of 1000 object categories. In this dataset, each typical category, such as "cat" or "strawberry", contains several hundred images.

Our logic behind the preference of the VGGNet architecture over more recent and powerful architectures such as GoogLeNet and ResNet is its suitability to represent texture patterns which is of the crucial importance in our segmentation and captioning task in which

we need to not only detect the objects but also important textured regions such as "sky", "field", "floor" and "wall". This unique property of the VGGNet architecture lies in three aspects. First, the convolutional filters of the VGGnet are so small that the edge patterns can still be observed in the fourth convolutional layer, while the object level patterns appear as late as the first component of the fifth convolutional layer. So, features that are captured by the first four layers are similar with the properties of textures which are larger than pixels and smaller than objects. Here we may argue that the features of the fifth convolutional layer are no longer suitable for representing texture, because layers in this stage contain object level patterns, which are too complex to describe textures. As our second reason, in contrast with the modern architectures, the number of the kernels in each layer of the VGGNet architecture is so high that textures can be represented effectively. Finally, the visualization of the features of the VGGNet is more interpretable than the other architectures that contain more sophisticated connection topology. This attribute is very useful when tried to tune the hyperparameters of our model during the training phase of the model and check its influence on the model discrimination strength.

It is worthwhile noting that the fully convolutional layers at the end of the pretrained VGGNet are responsible to calculate the final probabilities of the ImageNet classification labels and normally contain domain-specific data. Hence, they will be dropped during aforementioned knowledge transformation process.

Here, the pre-trained CNN (VGGNet) plays the role of a fixed feature generator that contains visual features of all the samples in the training dataset. It is clear that the backpropagation process can be used to adjust the network parameters to the contents of the newly observed data that is known as *finetuning* process. The output of such a feature extractor form the input to the next stage, automatic object detection and localization.

3.2.2 Deep Region Proposals

Since linguistic descriptions of the scene make numerous references to objects and regions of the input image, we need to transform generic input representation into sets of vectors that describe objects and regions. To this aim, the recognition module is designed to iden-

tify regions of interest in the scene and extract a fixed-size representation for each. Due to the layer arrangement of the previous convolutional feature extractor, if the input image has the size of $3 \times W \times H$, the generic input representation has the size of $512 \times \lfloor \frac{W}{16} \rfloor \times \lfloor \frac{H}{16} \rfloor$. Based on the proposed architecture for the Region Proposal Network (RPN) in [163], to generate region proposals, the RPN takes a 3×3 spatial window of the generic input representation which has the effective receptive field of 228 pixels on the input image and uses an extra 3×3 convolutional layer to convert the contents of the sliding window into a lower-dimensional feature vector of size 512. Then, this compressed representation will be fed into two sibling 1×1 convolutional layers where the first one plays the role of a box-regression (*reg*) that predicts multiple region proposals among the visual contents of the current sliding window. The number of proposed regions in each location is a hyperparameter of the RPN which is normally indicated as k .

The second 1×1 convolutional layer is a box-classifier (*cls*) that estimates the probability of being object or not object for each proposed region which can be implemented as a two-class *softmax* classifier. Since the RPN traverses the whole scope of the generic input representation using a sliding-window operation, the parameters of *reg* and *cls* layers are shared across all the locations of the generic input representation that in turn represents the whole area of the input image.

The box-regressor determines the proposed location of each region by 4 parameters denoting the dimensions of the bounding box and the coordinates of its center. So, it produces $4k$ outputs representing the proposed k boxes in each location, while the output size of the *cls* layer is $2k$ representing the probability of containing and not containing an object for each region, respectively.

To predict region proposals in different scales and aspect ratios, there were two traditional solutions. The first approach proposes the use of the input image in different scales during the training process which is known as the *pyramids of images* technique [80], while the second method is the application of multiple filters with different sizes on each location of the input image known as *pyramids of filters*. However, in RPN mechanism, the scale-invariance property of the region proposals is addressed by regressing normalized (having zero mean and unit variance) offsets from a set of translation-invariant anchors (see sec-

tion 2.10.3). Hereby, the model learns to predict 4 parameters of (t_x, t_y, t_w, t_h) to regress from the set of anchor parameters (x_a, y_a, w_a, h_a) to the set of output region parameters (x, y, w, h) via the following log-space scaling transforms:

$$x = x_a + t_x w_a \quad (3.1)$$

$$y = y_a + t_y h_a \quad (3.2)$$

$$w = w_a e^{t_w} \quad (3.3)$$

$$h = h_a e^{t_h} \quad (3.4)$$

During the regression process, each point of the generic input representation will be projected back into the original input image plane where each of k anchor boxes is associated with its own size and aspect ratio centered at the projected location. By default, 3 scales and 3 aspect ratios form 9 different anchors at each sliding position. This procedure, avoids the necessity of applying input images or filters in multiple scales and makes it possible to train and test the model in single-scale setting that speeds up the whole process of learning. The whole structure of the RPN is trained using the mini-batch gradient descent over a multi-task loss function:

$$L(\{p_i\}, \{t_i\}) = \sum_i L_{cls}(p_i, p_i^*) + \lambda \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (3.5)$$

where, i is the anchor index in the mini-batch, p_i is the predicted objectness probability of the anchor i , $t_i = (x, y, w, h)$ is the set of regression parameters for regressing the anchor i to the predicted bounding box of the model, p_i^* is the anchor sign which is 1 for positive anchors that have an IoU higher than 0.7 with a ground truth box and 0 for negative anchors if their IoU ratio is lower than 0.3 for all ground-truth boxes. t_i^* is also the set of regression parameters for regressing the anchor i to the ground truth bounding box and λ is a balancing parameters between *reg* and *cls* loss functions. The classification loss L_{cls} is the softmax log-loss over two classes of the object and its background, while the regression loss L_{reg} is the *smooth* L_1 distance function between predicted bounding box t_i and its corresponding

ground-truth t_i^* where the *smooth* L_1 function is defined as:

$$\text{smooth } L_1(d) = \begin{cases} 0.5d^2 & \text{if } |d| < 1 \\ |d| - 0.5 & \text{otherwise} \end{cases} \quad (3.6)$$

where d is the L_1 distance between the predicted bounding box t and its corresponding ground-truth t^* .

If an object is not detected during among the region proposals, there is no way to correctly classify it in the next phase. So, it is extremely important for the region proposals to have a high *recall* rate that is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Such a sensitivity can be achieved by generating very large numbers of proposals. Therefore, applying the proposed RPN mechanism over an input image with ordinary dimensions (e.g. 512×512) leads to a great number of region proposals with a high amount of redundancy. One possible solution for reduction of the processing complexity in the subsequent operations is to use a subsampling approach. For the training phase, the sampled mini-batch has the size of 256 where half of the samples are positive regions and the rest are negative ones. In the test time, the greedy *Non-Maximal Suppression* (NMS) [201] can be applied for subsampling.

The final step of the recognition process is done through an independent *recognition network* that consists of two fully connected (FC) layers of the VGGNet architecture with the size of 4096. The internal functionality of this network will be elaborated later in this section. Since FC layers require a fixed-size representation as their input, we need to convert region proposals of varying sizes and aspect ratios into a fixed-size representation. One possible conversion mechanism is the deployment of the *Region of Interest* (RoI) pooling mechanism [69] over the corresponding area for each region proposal on the generic input representation. If the corresponding area has the dimension of $x \times y$, the RoI pooling layer uses the max pooling operation to convert it into a small feature map of fixed size $X \times Y$. This can be done by dividing the corresponding area into $X \times Y$ grid of sub-windows of approximate size of $x/X \times y/Y$. The RoI pooling is a function of primitive convolutional features and region proposal parameters which is differentiable with respect to convolutional features but non-differentiable with respect to the region proposals. This is because

RoI operation uses region coordinates to crop the feature maps, but the *crop* operation is not differentiable with respect to the region coordinates. So, the gradients can not be back-propagated to the proposal coordinates.

A reliable alternative for RoI pooling mechanism is the fully-differentiable *bilinear interpolation* function [91, 95]. The input parameters of the bilinear interpolation are the generic input feature map U and a region proposal. During the interpolation process, the generic input feature map U will be interpolated to generate an output feature map V . After projection of the region proposal onto U , a sampling grid G of size $G_x \times G_y$ will be computed to associate each element of V with real-valued coordinates into U . This sampling grid forms a linear function of the proposal coordinates. Hence, the gradients can be backpropagated into region proposals as well.

So far, the whole structure of the RPN mechanism and the subsequent bilinear interpolation produces three different set of outputs. The first set consists of the region proposals, while the second one includes objectness scores of all the proposed regions. The third output is a huge tensor of size $B \times C \times G_x \times G_y$ where B is the number of sampled proposals and $C = 512$ is the unchanged number of channels in the generic input representation. Inside of the tensor, each matrix of size $C \times G_x \times G_y$ involves a compressed representation of the visual properties of one specific region that is called a *region feature*.

At the final stage of the recognition process, the aforementioned recognition network receives the flattened versions of the region feature matrices of size $C \times G_x \times G_y$ and generates their 4096-D counterparts named as *region codes*. The resulting region codes are also regularized by dropout [189] technique. The region codes corresponding to positive region proposals are packed in a shape of a matrix to provide visual information for the following RNN language model. As a side note, by embedding similar regression and classification modules at the end of the recognition network just like the arrangement we had at the end of the internal RPN, we obtain the second opportunity to refine region coordinates and objectness scores.

3.3 Automatic Image Description

As the two main branches of the artificial intelligence, Computer Vision (CV) and Natural Language Processing (NLP) have exploited a variety of their traditional and also machine learning techniques to develop new processing approaches. Based on the recently rising interest in the combinatorial data analysis that requires simultaneous process of both visual and linguistic data, the newly-born lingo-visual community take the advantages of the both scopes to tag web-based images, generate automatic video subtitles, instantly translate multi-language conversations and much more in the multimodal space of the social media. In this context, the automatic image description has emerged as the verbalization process of the most important visual and conceptual contents shown in a scene including but not limited to the objects and their characteristics, mood of the scene and the rational interactions between scene components.

Such a complex operation entails the comprehensive understanding of the scene as well as some complicated natural language processing steps that should be able to receive a visual representation and turn it into a valid and grammatically correct description. This process can be even more challenging when the description involves user-specific and specialized interpretations. It is worthwhile noting that, such a complete set of requirements turns the automatic image description also into an outstanding test bed for the assessment of the computer vision techniques.

During early attempts, the image description process was being investigated as the image annotation [185] which can be defined as the automatic assignment of some keywords to the digital image. By replacing keywords with some sentences that are able to describe not only the image objects but also their semantic relations and the scene background reality, automatic image description received more attention [95, 202].

3.3.1 Generative Captioning Models

Three different groups of models have been proposed to tackle image captioning problem. Models of the first group that are known as *generative models* exploit manually or automatically extracted visual features to provide a representation that is suitable to classify

the scene type, detect scene objects and their properties and discover presented relations and actions. Then the extracted visual information is turned into words and phrases that are used in a natural language generation (NLG) process to produce final descriptions. For example, in the pioneering work of Elliott and Keller [60], an explicit mapping mechanism called *Visual Dependency Representation* (VDR) is used to generate a *dependency graph* that symbolizes the spatial relations of the scene. Extracted relations are then will be matched with the *syntactic dependency tree* [139] of descriptions. VDR mechanisms have shown their efficiency not only in description production but also in image retrieval [61]. An important distinction parameter among this group is the type of the language model that they use to generate linguistic explanations. Within generative models, there are methods that employ *n-gram language models* [166] to generate a comprehensive description, but first they need to detect and form the attributes and dependencies between different components in triple structures [109, 118]. There are also some other approaches that prefer to use *maximum entropy language model* [64] due to its flexible word detectors.

The most recent architecture that is proposed to be used as a language model is the *recurrent neural network*. The ordinary work-flow in the RNN-based language models is to generate the caption one word a time where each word is conditioned on the context of the previously generated words and a set of image features [141, 100]. Due to the sequential word generation strategy, the first descriptive word has a huge impact on the quality of the descriptive caption. Therefore, it will be produced as the more likely caption for the whole input image which is the word (or the phrase) that indicates the name of the object which has obtained the highest objectness score during the preceding object recognition process. In this setting, the RNN is not only a language model but also a hybrid model that relies on both visual and linguistic attributes and generates linguistic descriptions of the image content through a multimodal embedding of the visual stimuli and the word representation in a joint vector space [140].

As a traditional approach for caption generation, a set of techniques are invented that use pre-defined *sentence templates* in which open slots in templates can be filled with the words that are explaining object properties and their rational relations. Such an information can be obtained via a *hidden Markov model* [161] that operates on a *gigaword external corpus*

[145] or a VDR [60].

There are also other schemes that employ sophisticated linguistic approaches to generate scene descriptions. In this category, the model proposed by Mitchell et al. [144] generates a huge amount of well-formed sub-sentences and rearrange them using a tree-substitution technique. At the same time, Kuznetsova et al. [110] proposed a training approach to induct tree-fragments from existing descriptions. Later, Ortiz et al. [152] proposed an automatic translation mechanism over VDR-sentence pairs along with an integer linear program to generate image descriptions.

The main drawback of generative models is their limitation on the variety of the output due to the complexity of the natural language generation. Moreover, their visual detection strength is an upper-bound to their description generation accuracy. In other words, if they fail to recognize an important visual component, generation of an informative description is not guaranteed.

3.3.2 Retrieval-based Captioning Approaches

Unfortunately, construction of the image descriptions within a natural language generation framework, causes some linguistic dilemmas that distort the focus from the fundamental image understanding problem and complicates the evaluation process of previously unseen descriptions [85]. As an alternative solution for the automatic image captioning, retrieval-based approaches formulated the captioning process as a retrieval problem in which the input image will be associated with the descriptions of a ranking set of previously described and visually similar images as candidate descriptions. These potentially useful descriptions can be combined in several ways to shape a proper description for the inquired input. Both the retrieval and ranking operations can be performed in a purely visual or a multimodal lingo-visual space.

Visual Retrieval Captioning

This group of retrieval-based captioning techniques, utilize a three-step procedure to detect visual similarities and transfer previous knowledge to unseen instances. In the first step,

some visual features will be used to form a visual representation space in which a similarity function can be defined. In the second stage, the similarity function is used to find training samples that are most similar to the input image and provide candidate descriptions. Finally, more information will be used to sort candidate descriptions and make use of them to generate the intended description.

For instance, Kuznetsova et al. [110] applied a bunch of detectors and classifiers on the input image to provide its semantic content. The method is then proposed a separate image retrieval step on each salient region of the input image to gather corresponding phrases from the candidate descriptions. The collected phrases are then passed into integer linear programming algorithm to generate final description.

In [75], authors employed color histograms, Gabor filters and Haar descriptors as well as SIFT and GIST local features to form a feature space. Next, they utilized a joint probability model to compare the textual data of the candidate descriptions with the visual content of the input image and determine the best that forms the final description.

Furthermore, Mason and Charniak [137] proposed a new generation process and formed the final description by considering only the textual information in the final alignment step where the conditional probability of observing a word among the final description is calculated by a non-parametric density estimator that utilizes previously extracted candidate descriptions. The final description is then extracted via two different extractive summarization techniques [146, 97].

As some deep learning approaches in this category, Yagcioglu et al. [205] proposed an average query expansion technique in which required features are extracted from a VGGNet [184]. Then the original query is expanded as the average of the distributed representations of candidate captions that are weighted based on their similarity to the input image. Similarly, Devlin et al. [55] applied filters of a trained CNN as the global image descriptor followed by the k-nearest-neighbors classifier to find similar training images to the input image.

Lingo-Visual Retrieval Captioning

In the training phase of the third group of the automatic image captioning models, both sets of the visual and linguistic features will be transformed into a multimodal feature space. At the test time, the generated feature space will be used to extract and (in more advanced models) generate new captions for the test samples which is known as the *cross-model retrieval*. So, models of this group are able to retrieve the most accurate descriptions from a large previously-formed pool of captions. In addition, such a multimodal feature space provides this opportunity to handle the reverse problem of retrieving the best fit image for the given textual explanation.

As mentioned in Girshick et al. [65], a joint meaning space is a proper basis to match knowledge of different domains. Hence, authors in [85] took the advantage of a *Kernelized Canonical Correlation Analysis* (KCCA) [87] to transfer images and their corresponding descriptions into a higher-order space in order to discover hidden relations in between. Unfortunately, in KCCA mechanism, it is necessary to keep the kernel computations in the system memory until the end of the training process that makes it impracticable for large datasets.

In [186], neural architectures are used to provide vector representation for both visual and textual information. In case of the linguistic information, a *Dependency Tree RNN* (DT-RNN) is utilized to generate compositional sentence vectors over word order and syntactic differences that leads to a 50-dimensional word embedding system. The obtained representations are first learned in their own feature space and then will be converted into the multimodal feature space using a max-margin objective function that enforces matched pairs of image and description to have a high inner product than misaligned pairs, by a margin. Instead of directly mapping description and images, in [101] smaller units of words and their visual counterparts are transferred into the common feature space to enhance model accuracy.

One step further, Kiros et al. [103] proposed an encoder-decoder package to not only rank but also generate descriptions where the encoder part of the model is composed of a CNN-LSTM architecture that constructs a joint feature space and the decoder component

is a *Structure-Content Neural Language Model* (SC-NLM) that applies the resulting multimodal representation to produce the new set of explanations. Later, Donahue et al. [57] proposed a model that encapsulates a copy of the static image and the previous word of the description as the input package of an LSTM cascade to generate next part of the description.

As one of the most successful approaches in this context, Karpathy and Fei-Fei [100] assumed that different parts of the description refer to specific but unknown regions of the visual input. So, they proposed a model to infer the alignment between parts of sentence and regions of the image. Their architecture consists of three main parts. The first module is a convolutional architecture [70] that is responsible to generate region representations in a *h-dimensional* space, similar to our hybrid feature extraction and localization unit that is explained in section 3.1. The second component is a *bi-directional RNN* (BRNN) [140] that is responsible to produce a representation for each word of the vocabulary and its context from both sides in the same *h-dimensional* space. Due to overfitting concern, the proposed BRNN exploits the word embedding of an unsupervised method such as *Skip-gram model* [142] as the initial values. The last part of their model is a max-margin loss function that aligns two modalities.

3.3.3 Recurrent Language Model

All the captioning approaches that deal with this problem as a retrieval query, try to find an appropriate explanation for the input image by an iterative search over numerous human-written candidate descriptions that is obviously inefficient. Despite the aforementioned advances in this category, such a framework is still suffering from some limitations. First, they require a huge amount of diverse training data in the form of image-description. In addition, these methods are restricted to a fixed set of descriptions and the models are not able to generalize well to new concepts. Moreover, the retrieval framework is not the way that humans exploit to describe images.

In contrast to the retrieval techniques, here the ultimate goal is to design a model that is able to generate a variable-sized description for the given input where each training sample of

the model consists of an image and its description. Among the NLP community, a language model is defined as a technique that aims to provide the probability distribution over word sequences $x_1 \dots x_T$ in the form of a joint probability distribution $\prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1})$ which is equivalent to compute the conditional probability of a word based on the previous words in the sequence.

While the traditional models of *n-grams* [118, 166] employ the combination of a lookup table and smoothing techniques to condition a word on the previous n words, a recurrent neural network language model [135, 41, 57, 200, 64] is a more flexible structure that is able to capture longer dependencies between the words. In addition, such a model is potentially suited to be also conditioned on the image information and form a multimodal RNN language model as the core component of an automatic caption generation system.

During training, the given image is first processed by a CNN to extract the feature vector I that summarizes the whole visual content. This feature vector is then passed to the language model as the visual input of the RNN indicated by (x_{-1}) . The next item of the input sequence is the special START token that is symbolized by x_0 followed by the sequence of words in the training description $(x_1 \dots x_T)$.

In the framework of a language model, each sentence is a sequence of words drawn from a pre-determined vocabulary V . So, the primary encoding mechanism is to represent each word with a *one-hot* vector that is all zero except for a unique 1 at the index of that specific word in the vocabulary. Hence, $\mathbb{I}_t \in R^{|V|}$ is the one-hot encoding of the t -th word in a vocabulary. In addition, the distributed representation of words in a vector space encodes many linguistic regularities and patterns that help the learning algorithms to achieve better performance [142]. Thus, as the complementary encoding step, the input vectors $(x_1 \dots x_T)$ will be encoded by the linear projection of one-hot input vectors using a word embedding matrix W_w that can be learn via backpropagation:

$$x_t = W_w \mathbb{I}_t \tag{3.7}$$

In the case of the data scarcity and/or overfitting concern, the word embedding matrix W_w can be set by an unsupervised method such as Skip-gram model [142] that is fixed during

the training process.

The Recurrent language model generates sequences of hidden states ($h_1 \dots h_T$) and output vectors ($y_1 \dots y_T$) as the results of the following recurrence formula:

$$\bar{I} = W_{hi}[CNN_{\theta}(I)] \quad (3.8)$$

$$h_t = \begin{cases} ReLU(W_{hx}x_t + W_{hh}h_{t-1} + b_h + \bar{I}) & \text{if } t = 1 \\ ReLU(W_{hx}x_t + W_{hh}h_{t-1} + b_h) & \text{if } t > 1 \end{cases} \quad (3.9)$$

$$y_t = softmax(W_{oh}h_t + b_o) \quad (3.10)$$

Here, \bar{I} is the image content that is provided to the language model, h_t is the hidden state of the RNN at the time step t and y_t is the output vector of the network at the same time step.

The output vector y_t is of size $|V + 1|$ and holds the unnormalized log probabilities of the words in the dictionary and the additional special token END that is expected to be produced at the end of each generated description. A related point to consider is that the hidden state is initialize by a zero vector h_0 . All the weight and bias matrices are the learning parameters of the network.

It is worthwhile noting that the visual information of each region is exposed to the language model once at the first time step of the training process and the model exploits a little part of its internal dynamics to form a local identity connection that spreads the visual content to all the time steps. Moreover, it is empirically verified that feeding the image at each time step as an extra input yields inferior results, as the network can explicitly exploit noise in the image and overfits more [200]. Having this strategy, the language model is able to not only remember the visual information in its hidden state, but also employ its internal dynamics to track context information and generate rich descriptions. The number of neurons in the hidden state of the RNN language model is a hyper parameter that should keep a balance between the computational complexity of the model and the model performance. At the training stage of our hybrid architecture, x_{-1} is the region code that is provided by the recognition network. x_0 is the special START token and the expected outcome for y_1 is the first word in the ground truth caption of that specific region that is provided by the

training dataset. At the next time step, the network input x_2 is the word vector of the first word in the ground truth caption and expected output y_2 is the second word of the given description. We iterate this process until the time step in which the network generates the special END token. The objective function can be any function that is able to maximize the log probability of the targets such as softmax classifier or the cross-entropy loss.

At the test time, after feeding the network with the START token, the most likely next token will be sampled to form the input of the next time step. This process is also iterated until the generation of the END token. The combination of the previously explained recognition module including the pretrained CNN, RPN, bilinear interpolation mechanism and the fully connected recognition network, and the aforementioned language model forms the *Recognition and Captioning* (ReCap) module of our hybrid model which is shown in Fig 3-2

3.4 Deep Interactive Segmentation

With the growing popularity of interactive devices such as smart phones and tablets, interactive image processing attracts more attention. Interactive segmentation offers a pixel-wise classification based on user priorities. Among traditional approaches for interactive segmentation, stroke-based techniques [119, 199] are often combined with graph cut methods. In these approaches, an energy function based on region/boundary division is optimized to find the segmentation result. Alternative approaches include random walks [73] and geodesics [48], mostly rely on low-level features such as color, texture and shape information. These types of attributes can be difficult to apply when the image appearance is complicated due to complex lighting conditions or existence of intricate textures. Recently, deep learning models have been used for interactive segmentation where the information of the image will be considered in higher semantic levels. To this end, FCNs as the standard frameworks for the pixel-wise end-to-end learning tasks, have been applied. The main difficulty in the training process of the deep architectures is the problem of vanishing gradients [156]. The Deep residual learning network (ResNet) [81] seems to be a highly promising attempt to solve this problem in a simple way by adding identity mapping con-

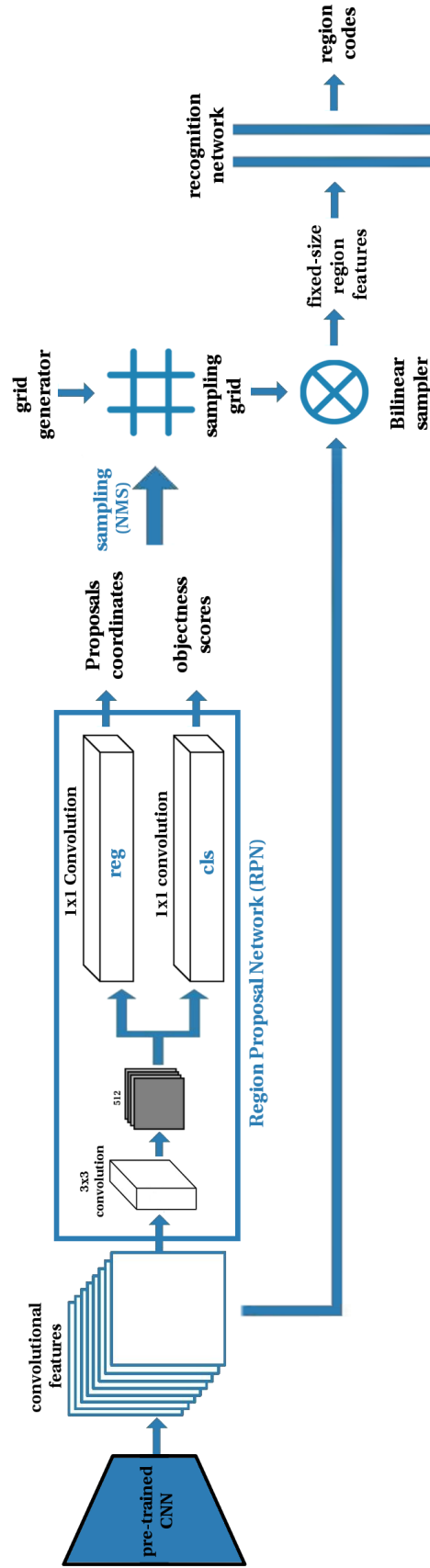


Figure 3-2: The recognizer part of the proposed Recognition and Captioning (ReCap) module [28].

nections to the network blocks in order to facilitate the gradient flow through the network. So, fully convolutional versions of the ResNet lead to excellent results in segmentation tasks [40, 58]. Recently, the Densely Connected Convolutional Networks (densenets) [88] and their fully convolutional versions (fcdensenets) [93] made it possible to train very deep structures with more than 1000 layers without any sign of vanishing gradients while using a less number of parameters in comparison with previous deep architecture.

In most of the previous researches, detected objects are determined by locating bounding boxes around them. Although this notation is able to simplify the detection process by converting the segmentation task to a regression problem, such an output is less informative when dealing with geometrical properties of the objects. As more illustrative visual recognition techniques, semantic segmentation [127, 120] aims to assign a label to each pixel of the image where the labels can be class-aware or instance-aware, while the interactive image segmentation [20, 119, 199, 73, 48] tries to incorporate the segmentation task with the user preferences. In reality, it sounds reasonable that human users may have a more restricted area of interest than the entire scope of the scene. Thus, the interactive region segmentation can be defined as a binary segmentation problem that separates the User Intended Region (UIR) as the foreground from the other parts of the scene as the background.

3.4.1 User Interaction Imitation

During the interactive segmentation process, the user will be asked to provide some general information about the position of the intended region. The requested information consists of some positive and negative seeds as depicted in Fig. 3-3 which are equivalent to internal and external points of the UIR, respectively. Since the manual collection of such interactions is very expensive and time consuming, a better strategy is to imitate these interactions synthetically. After automatic generation of positive and negative clicks for each training sample, each cluster of seeds (positive cluster and negative cluster) will be used to construct a Voronoi diagram.

In each cluster, we denote every seed by s_k , $k = 1, \dots, n$. The value of pixel $v_{i,j}$ of the

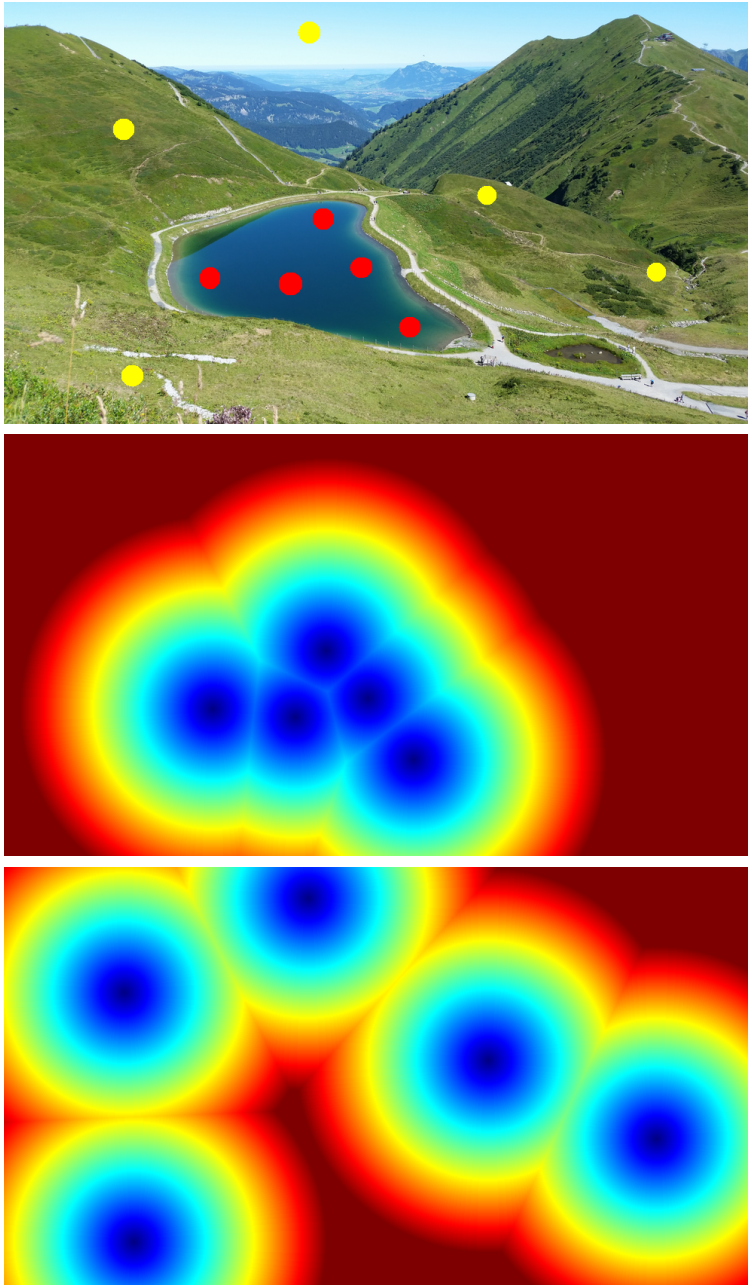


Figure 3-3: Input image including positive and negative clicks (up), and obtained positive (middle) and negative (bottom) Voronoi diagrams [28].

positive/negative Voronoi diagram will be calculated by

$$v_{i,j} := \min\{D_1^{i,j}, D_2^{i,j}, \dots, D_n^{i,j}\} \quad (3.11)$$

where $D_k^{i,j}$ is the Euclidean distance of the pixel $v_{i,j}$ to the seed s_k . To summarize, the value of each pixel in the positive/negative Voronoi diagram is the Euclidean distance of that pixel to the nearest positive/negative seed. To emulate the natural behavior of a human user in choosing the locations of positive and negative clicks, inside and outside of the intended region, the following conditions should be satisfied:

First, we should note that the human user does not usually select a same location for two different clicks. So, every pair of seeds in each cluster should preserve a predefined distance from each other:

$$\exists d_1 \in \mathbb{R}^+ : \forall s_i, s_j \in S, \|s_i - s_j\|_2 > d_1 \quad (3.12)$$

In addition, the human user normally does not click over the locations that are too close to the boundaries of the intended region. Therefore, all the seeds of each cluster should preserve a minimum distance from boundary pixels of the UIR ($\partial(UIR)$):

$$\exists d_2 \in \mathbb{R}^+ : \forall s_i \in S, u \in \partial(UIR), \|s_i - u\|_2 > d_2 \quad (3.13)$$

Here S denotes all the seeds of a cluster. Recently, Xu et al. [203] proposed some strategies for synthetic generation of user interactions. They considered random generation of positive clicks inside the UIR while three distinct cluster of negative clicks are chosen as: 1) random background pixels with a certain distance to the UIR, 2) a point cloud inside the negative (not intended) objects and 3) a uniform set of points surrounding the UIR. Although the first two strategies do not obey the common patterns of the user clicks, they aim to add some randomness to the training data, while the third strategy provides some geometrical information about the UIR which can be learned by gradient descend approaches.

3.4.2 Morphological Cortex Detection (MCD).

While the inside of the UIR may be quite small, the background region is often large enough to provide useful geometric information about the UIR. Consequently, it seems beneficial to generate negative seeds that surround the UIR uniformly. To provide an efficient implementation for such a selection, we introduce our Morphological Cortex Detection (MCD) technique that is computationally efficient. To implement this idea, a 1-pixel-wide shape of the geometric boundary of the UIR is extracted by performing the morphological dilation operation on the binary mask of the UIR and subtracting the original mask from the dilation result. In the next step, the obtained boundary shape of the UIR will be completely traversed using a 3×3 window to transfer all the boundary points' coordinates into a 1-D array in which the requested negative seeds can be selected uniformly. As the result of the MCD process, a uniform set of negative seeds will be obtained that represents the cortex of the UIR perfectly. In addition, by the random selection of the number of the recursive executions of the dilation operation, our MCD approach is able to generate UIR cortex at different distances which is able to add more randomness to the generation of synthetic clicks. The visual illustration of this technique is shown in Fig. 3-4. During our experiments, positive clicks are determined randomly inside the UIR, while negative seeds are generated in three different clusters using 1) our MCD technique, 2) a random selection of the background pixels, and 3) a random selection inside a negative object (if any). The combination of one positive and three negative clusters can be used to generate up to 6 *training pairs* for each intended region in the training data.

3.4.3 Fully Convolutional Interactive Segmentation Networks

Our hybrid model receives an input image as well as user interactions in the form of positive/negative clicks and provides a seamless framework to generate accurate segmentation as well as expressive description of the UIR. In a dedicated preprocessing step, an effective morphological technique is proposed to provide a huge amount of training samples in the form of synthetic user interactions. Then, each cluster of positive/negative seeds will be transformed into separate Voronoi diagrams as shown in Fig. 3-3. As the next step, one

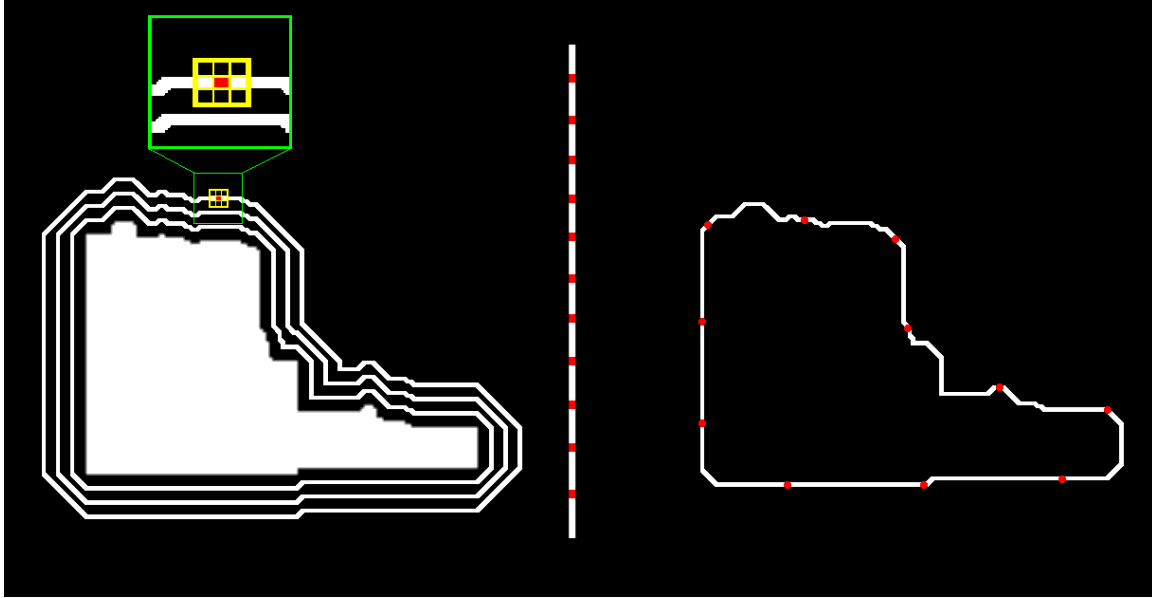


Figure 3-4: Our proposed Morphological Cortex Detection (MCD) technique [28].

of the FCN, LFCN or FCDenseNet architectures is applied as the deep interactive segmentation module. The structural details of these architectures will be elaborated later in this section. In parallel, the ReCap module is utilized to obtain region proposals along with their captions. In the fusion step, a heuristic method will be provided to combine results of the localization, segmentation and captioning procedures. In the following, we will investigate all the steps of our model in detail.

As the interactive segmentation module, we applied three different fully convolutional architectures in three different version of our hybrid model:

- (i) The traditional FCN proposed in [131].
- (ii) Our special version of FCN where last two convolutional layers of the traditional FCN are replaced by three convolutional layers with kernel sizes of 7, 5 and 3 named as Lyncean Fully Convolutional Network (LFCN).
- (iii) The state-of-the-art FCDenseNet [88, 93] that contains 103 convolutional layers.

The impact of the alternation in LFCN is the gradual growth of the receptive field. Our experiments show that this property is able to improve network recognition accuracy. By

a proper use of zero padding, all the extended convolutional layers have the same output size. The FCDenseNet architecture is a newly proposed architecture in which each layer receives not only the output feature maps of the previous layer but also the aggregated feature maps of all the previous layers via direct connections. This property mitigates the vanishing gradient problem, facilitates feature propagation and decreases the number of the network parameters (see section 2.6.7).

3.4.4 Fusion Approach

In order to attach a proper linguistic commentary to the output of the interactive segmentation module, the segmentation result should be enriched with a selective output of the ReCap module. As a reminder, the ReCap module determines the approximate location of each identified object of the input image by specifying a bounding box around it. These bounding boxes are also known as region proposals. Each region proposal has a confidence score as the probability of the existence of an object in it. After a descending sort of all the confidence scores, locations of the most likely objects can be obtained from the top-ranked region proposals. In all our experiments, we picked up top-10 region proposals with the highest confidence scores as the best candidates for the fusion with the segmentation result. The last unit of ReCap module is a RNN-based language model that provides a linguistic explanation for the visual properties of each region proposal. In the final step of the fusion process, we compare the location of the interactive segmentation result with the locations of the top-10 bounding boxes using the well-known *Intersection over Union* (IoU) metric to find the best match bounding box. Finally, the caption of the best match bounding box will be used as the most appropriate choice to describe the output of the interactive segmentation module. The general view of the proposed model including both interactive segmentation and ReCap modules is illustrated in Fig. 3-5.

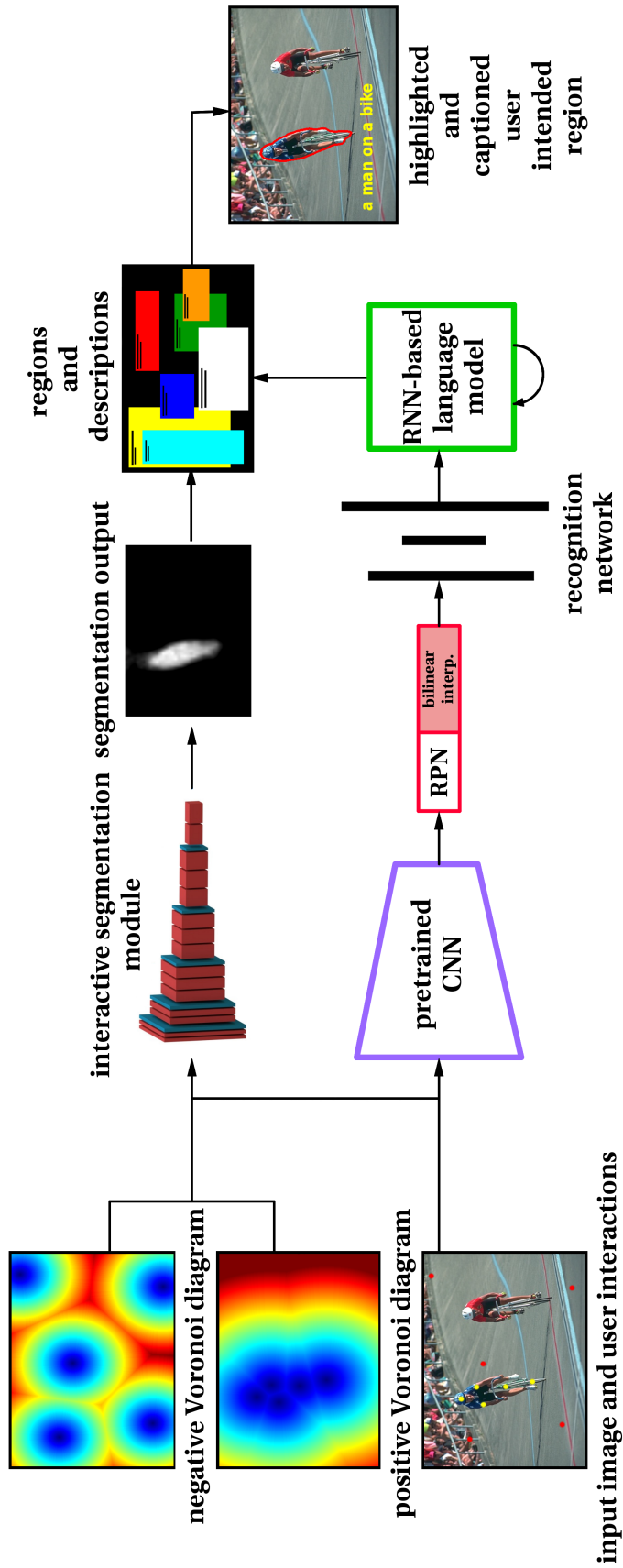


Figure 3-5: Architecture of the proposed deep interactive region segmentation and captioning model [28].

3.5 Experimental Results

3.5.1 Datasets

All the components of our ReCap module including CNN, RPN, bilinear pooling mechanism, recognition network and the RNN language model are trained jointly using the Visual Genome (VG) region captions dataset [106]. The dataset includes 108,077 images and 4,297,502 region descriptions with the average of 40 captions per image that are collected on Amazon Mechanical Turk.

For the fine tuning of FCN, LFCN and FCDenseNet we used the PASCAL VOC 2012 segmentation dataset [62]. The dataset includes 1464 images for training and 1449 images for validation that are distributed in 20 different classes. We used the whole bunch of these samples to generate our special training pairs in the preprocessing step. For the final validation of the model as well as its comparison with state-of-the-art interactive segmentation, we utilized different well-known segmentation benchmarks including Alpha Matting [165], Berkeley segmentation dataset (BSDS500) [136], Weizmann segmentation evaluation database [13], image object segmentation visual quality evaluation database (MOS) [183] and VOC validation subset.

3.5.2 Preprocessing

For a better use of the Visual Genome dataset as proposed in [95], we converted words that appear less than 10 times into the special <UNK> token, resulting a vocabulary of 9,873 words. We also discarded all the proposed captions with more than 10 words and the images that have less than 15 or more than 45 captions to decrease the variation in the number of regions per image that yields into 84,742 images. Each of our test and validation data contains 7371 images, while the training data includes 70,000 samples. During our preprocessing step, we have also merged strongly overlapping boxes into single boxes with several reference captions. To do so, for each image, we iteratively determined the box with the highest number of overlapping boxes (specified by the IoU value of 0.7 or higher between two bounding boxes) and merge these together by taking the mean. To prevent an

excessive merging process, the resulting bounding box of each iteration will be excluded in further merging iterations.

To generate the maximum possible number of training pairs for the interactive segmentation module, we produced positive and negative Voronoi diagrams with respect to each object that is labeled in the VOC dataset. The positive seeds are selected randomly inside each object while the MCD approach is used to generate three distinct sets of negative seeds with different distances from the intended object. In the last step, each combination of positive/negative Voronoi diagrams along with the original input image, forms a unique training pair. This leads to production of 97,055 interaction patterns. We preserved 7,055 instances for the test and used the rest as the training data.

3.5.3 Training Process of the ReCap and Interactive Segmentation Modules

During the training phase of the ReCap module, the ground truth consists of positive boxes and descriptions. The module predicts locations and the objectness scores of the sampled regions twice: within the RPN and again in the recognition network. We use binary logistic losses for the confidences trained on sampled positive and negative regions, while as mentioned before, a smooth L1 loss is used for the training of the box regressor. The final term in the loss function is a cross-entropy term at every time-step of the language model. All the loss functions are normalized by the batch size. We initialize all the weights of the ReCap module randomly using a Gaussian distribution with standard deviation of 0.01 except the CNN that is pretrained on the ImageNet object classification task. During the optimization, we used the stochastic gradient descent approach with the momentum of 0.9 to train the weights of the convolutional network, and Adam [102] to train the other components of the model. We use a learning rate of 1e-6 and set $\beta_1 = 0.9$ and $\beta_2 = 0.99$. We begin fine-tuning the layers of the CNN after 1 epoch, and for efficiency we do not fine-tune the first four convolutional layers of the pretrained CNN.

To reach the best quality for the interactive segmentation, FCN and LFCN are trained in three different levels of granularity as proposed in [131]: X32s, X16s and X8s. RGB chan-

nels of the input image are concatenated with the corresponding Voronoi diagrams to form a training instance. Consequently, the first (input) layer of the interactive segmentation module contains five channels. During the network initialization, the RGB-related channels will be initialized by the parameters of the pretrained FCN [131] on ImageNet [172] classification task. For two extra channels that are associated with Voronoi diagrams, the zero initialization is the best choice as also reported in [203]. Learning parameters of the finer networks should be initialized from the coarser one. The global learning rates of the FCN and LFCN networks are $1e-8$ for X32s, $1e-10$ for X16s and $1e-12$ for X8s granularities. In LFCN architecture, the extended convolutional layers exploit one hundred times bigger learning rates. The learning policy is fixed and we used the weight decay of $5e-3$. Both networks are trained for 18 epochs.

As our third choice for the interactive segmentation module, we utilized the FCDenseNet [93] containing 103 convolutional layers where the down-sampling path consists of 5 dense blocks with 4, 5, 7, 10 and 12 bottleneck layers [88] respectively. The middle block (the connector of the down-sampling and up-sampling paths) includes 15 bottleneck layers while the up-sampling path has 5 dense blocks with 12, 10, 7, 5 and 4 bottleneck layers. The first convolutional layer of the network (before down-sampling path) has 48 filters and the growth rate of the network is 16. The final convolutional layer (after up-sampling path) has 2 filters, one for foreground and the other for the background area that is followed by a Softmax classifier. We trained our FCDenseNet for 15 epochs with the learning rate and the weight decay of $1e-4$, while the batch size was 5. All the networks are trained on random crops and horizontal flips of the training data with the size of 256×256 .

3.5.4 Segmentation Accuracy Metrics

There are several measures to evaluate the segmentation accuracy. As mentioned before, the segmentation part of our hybrid process can be considered as a binary segmentation where the classes are limited to foreground (UIR) and the rest of the image as background. So, we employed the binary interpretation of the semantic segmentation metrics that are proposed by Long et al. [131]:

- **Pixel Accuracy (Pixel Acc.):**

This measure represents the proportion of the correctly classified foreground (C_f) and background (C_b) pixels (true positive rates) to the total number of ground truth pixels in foreground (F) and background (B).

$$\frac{C_f + C_b}{F + B} \quad (3.14)$$

Unfortunately, this metric can be easily influenced by the class imbalance. Hence, high pixel accuracy does not necessarily mean that the accuracy is acceptable when one of the classes is too small or too large.

- **Mean Pixel Accuracy (Mean Acc.):**

This measure is computed as the mean of the separate foreground and background pixel accuracies:

$$\frac{\frac{C_f}{F} + \frac{C_b}{B}}{2} \quad (3.15)$$

This metric alleviates the imbalance problem but can be still misleading. For example, when the great majority of pixels belong to the background, a method that predicts all the pixels as background can pretend a good performance.

- **Mean Intersection over Union (Mean IoU):**

Intersection over union is the matching ratio between the result of object localization process and the corresponding ground truth label. This metric is the average of the computed intersection over union for foreground and background areas:

$$\frac{\frac{C_f}{C_f + FP_f + FN_f} + \frac{C_b}{C_b + FP_b + FN_b}}{2} \quad (3.16)$$

Here, FP and FN indicate false positive and false negative rates.

3.5.5 Test of Localization Accuracy

In our first experiment, we evaluated our model using a random subset of unseen test samples of our previously generated validation pairs. The response of the model to some in-

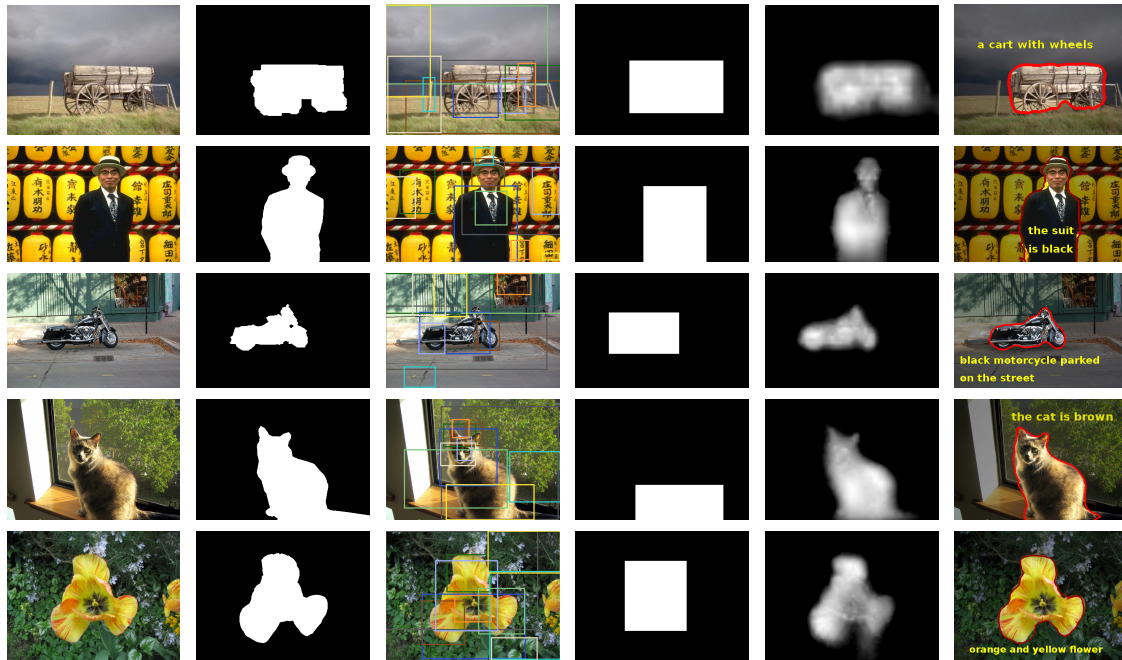


Figure 3-6: Operating progress of our proposed hybrid model, from left to right: input images obtained from PASCAL VOC 2012 dataset [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.

stances is shown in Fig. 3-6, while Fig. 3-11 to Fig. 3-15 provide more experimental results at the end of the chapter. It can be noticed that the output of our approach achieves a considerable rate of accuracy regarding the similarity of the model output with the corresponding ground truth. Furthermore, the confusing output of the dense image captioning is replaced with an illustrative outcome where the segmented UIR and its description are easily distinguishable. Fig. 3-7 (top diagram) presents a comparison between the localization accuracy of the proposed method where the interactive segmentation module is LFCN8s and the internal RPN of the ReCap module in terms of the obtained IoU for the samples presented in Fig. 3-6. As illustrated, our model provides a significant improvement regarding the localization accuracy by the replacement of the best match bounding box with the segmented UIR.

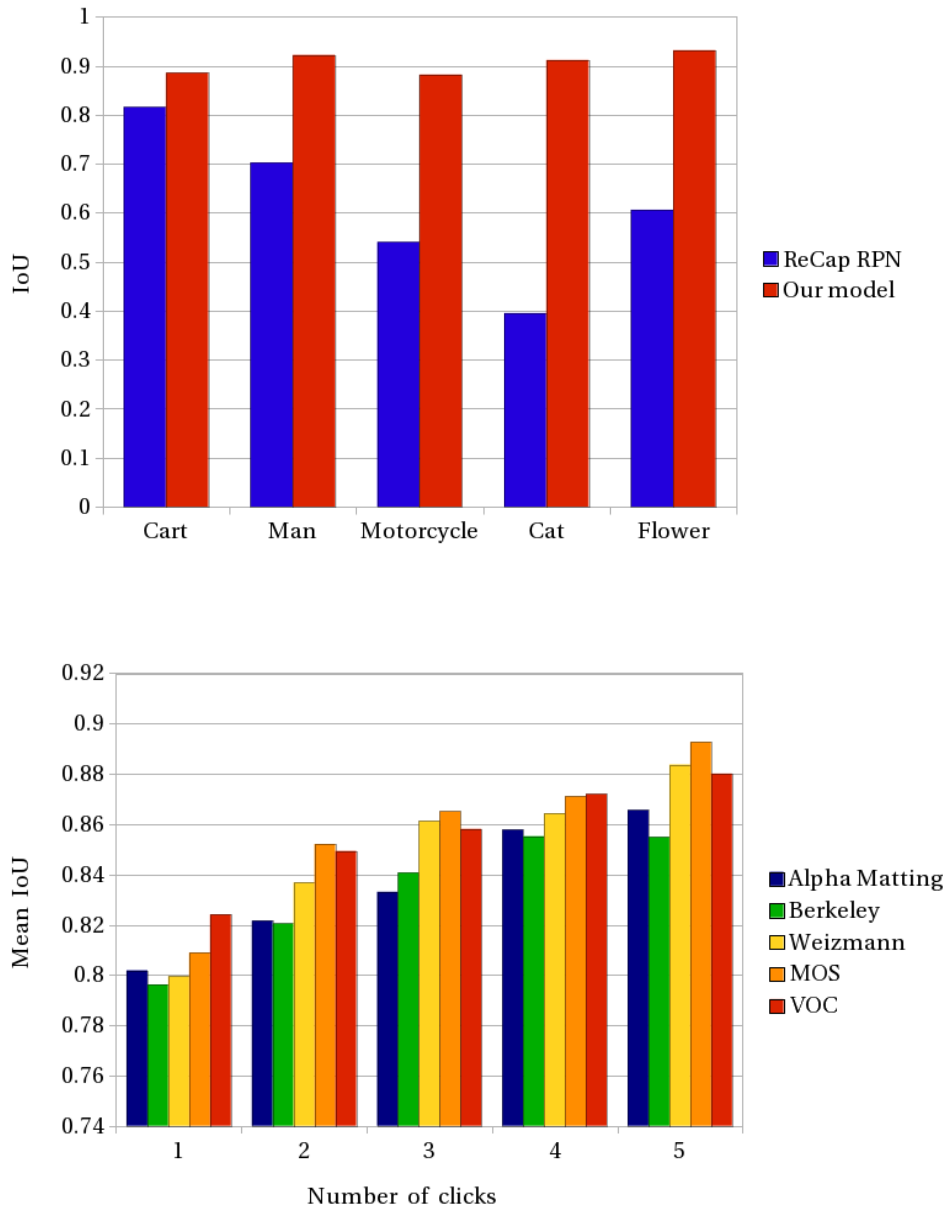
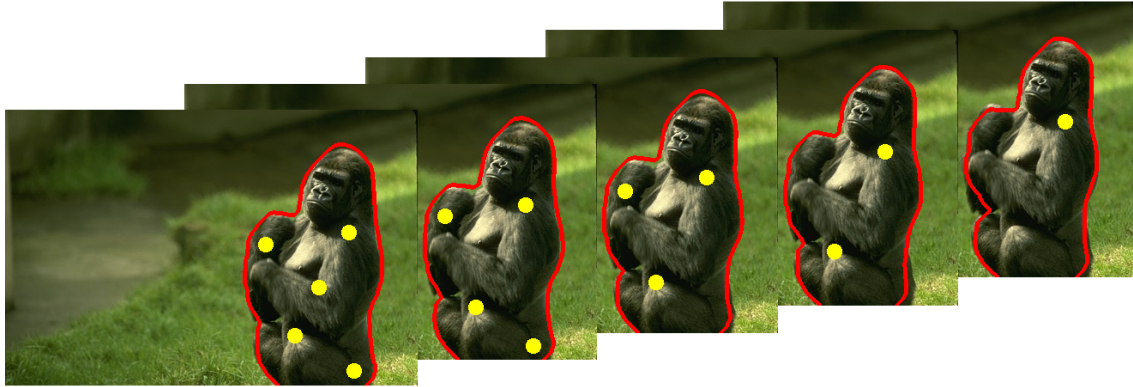


Figure 3-7: Localization accuracy comparison between our model (LFCN8s) and the internal region proposal network (RPN) of the ReCap module (up), and mean IoU accuracy of our proposed model (LFCN8s) on several segmentation benchmarks against different number of clicks (bottom) [28].



Number of clicks	IoU
One	0.8895
Two	0.9141
Three	0.9227
Four	0.9406
Five	0.9485

Figure 3-8: Model sensitivity analysis against number of clicks on a sample input image obtained from PASCAL VOC 2012 dataset [62] (up), and its associated IoU rates (bottom). Here, the IoU measure is computed between the segmentation result and the ground truth mask of the object.

3.5.6 Sensitivity Analysis

In this part, we analyzed the output quality of the interactive segmentation module against the number of user interactions. As it is shown in Fig. 3-8, although IoU can be improved by applying more user interactions that facilitate boundary detection, our model still provides very good results even by minimum number of clicks. We also measured mean IoU accuracy of the proposed model for five different datasets as illustrated in Fig. 3-7 (right diagram). The experiments confirm satisfying performance of our model in the case of a low number of user clicks. We conjecture that this noticeable property of our approach makes it convenient to be applied in real-world applications. During our experiments, the proposed method clearly achieves a satisfying segmentation outcome with just one click cf. Fig. 3-8.

3.5.7 Quantitative Comparison of the Segmentation Quality

In this part we performed an extensive evaluation on segmentation capabilities of the proposed method versus some prevalent segmentation techniques such as Geodesic Matting (GM) [20], GrowCut [199], Grabcut [169], Boykov Jolly (BJ) interactive graph cuts [31], Geodesic Star Convexity (GSC) and Geodesic Star Convexity with sequential constraints (GSCSEQ) [74], Random Walker (RW) segmentation [73], Shortest Path-based interactive segmentation (SP) [96] and Matching Attributed Relational Graphs (MARG) [148]. In all the experiments we generated five positive and five negative clicks randomly. For some of the approaches where the user interactions were defined as points or scribbles, we determined click positions with five-pixel-wide circles. Tables 3.1 to 3.5 present quantitative results of these experiments that confirm the superiority of our approach over all the other techniques on different segmentation benchmarks. Due to measure quality discussions in section 3.5.4, less reliable accuracy rates are presented in gray.

As a qualitative comparison, Fig. 3-9 represents final segmentation results of the methods in Table 3.1 on two different test samples. As it can be seen, our deep interactive segmentation modules provide the most accurate segmentation results with respect to human-like interpretation of the scene. A related point to consider is that in all the experiments the number of the user interactions are limited to two positive and two negative clicks.

3.5.8 Dense Interactive Region Captioning

In the final part of our experiments, we verified the ability of the proposed model to caption several regions of an image. In Fig. 3-10, we show the result of such an experiment where multiple objects in different scales are detected via user interactions and described properly.

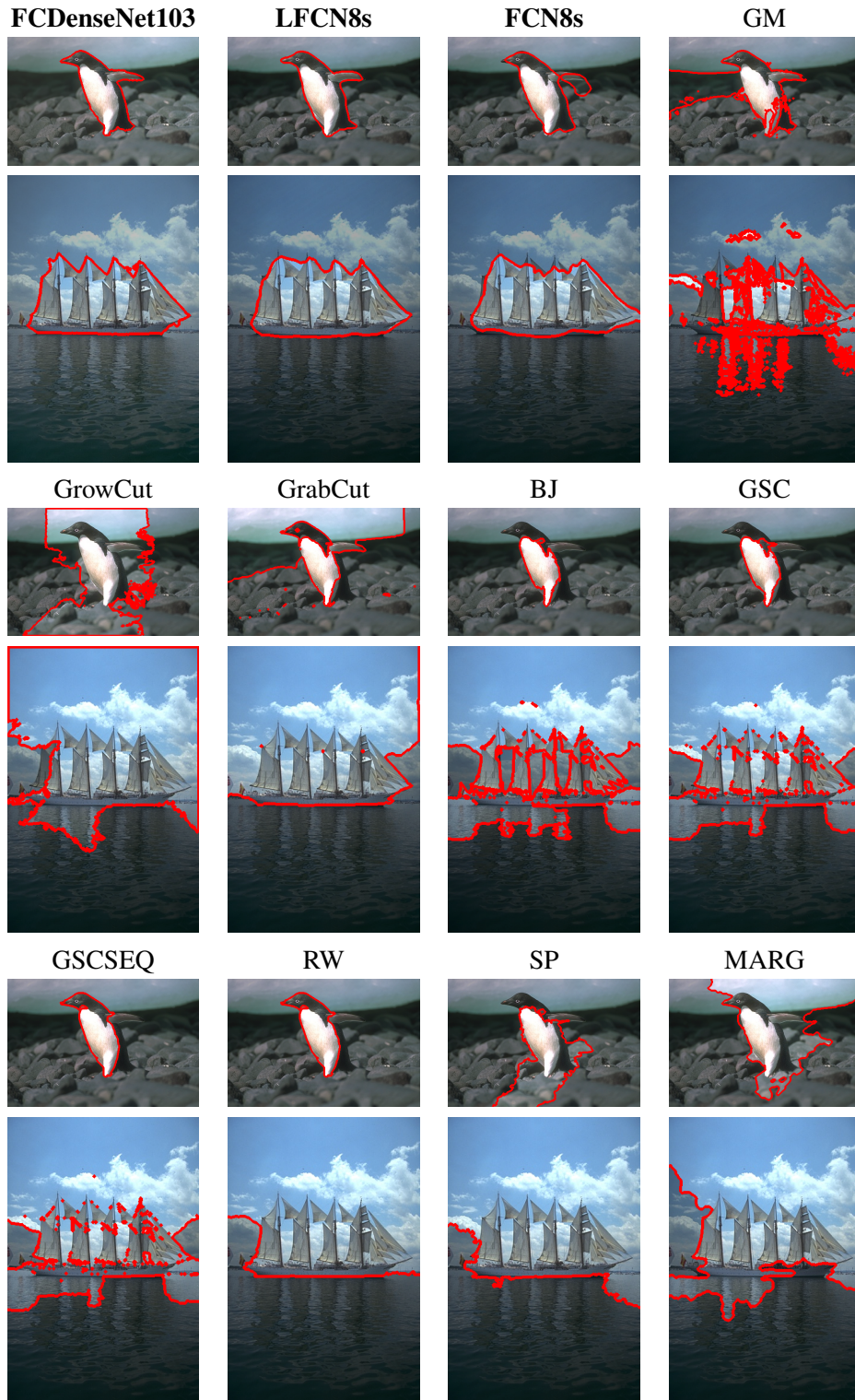


Figure 3-9: Segmentation quality comparison between our three proposed interactive segmentation modules and other interactive segmentation techniques. Our method clearly provides more abstract region and object understanding. Input images are obtained from PASCAL VOC 2012 dataset [62]

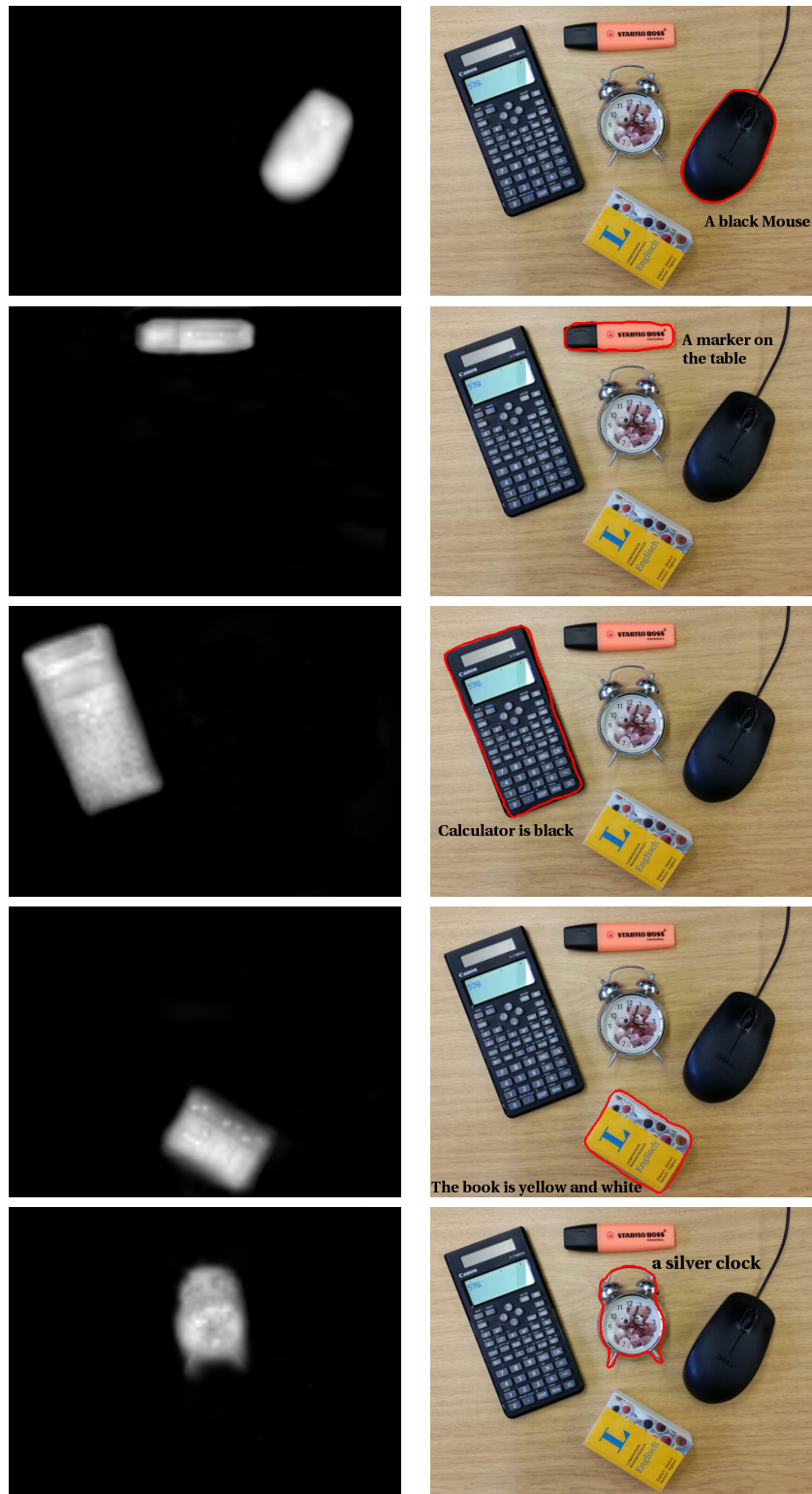


Figure 3-10: Output probability map (left) and the final result (right) of our approach for different regions of an image in response of different user interactions [28].

	Pixel Acc.	Mean Acc.	Mean IoU
GM [20]	0.8237	0.7645	0.6098
GrowCut [199]	0.6639	0.7603	0.4520
GrabCut [169]	0.6614	0.7582	0.4511
BJ [31]	0.8138	0.8292	0.6488
GSC [74]	0.8240	0.8465	0.6624
GSCSEQ	0.8275	0.8425	0.6654
RW [73]	0.8691	0.8046	0.6917
SP [96]	0.7838	0.8405	0.6121
MARG [148]	0.8067	0.6516	0.6180
FCN8s	0.9379	0.8766	0.8352
LFCN8s	0.9597	0.8989	0.8549
FCDenseNet103	0.9805	0.9473	0.9234

Table 3.1: Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on Berkeley (BDS500) dataset [136].

	Pixel Acc.	Mean Acc.	Mean IoU
GM [20]	0.8328	0.7906	0.6615
GrowCut [199]	0.6847	0.7025	0.4795
GrabCut [169]	0.6896	0.7103	0.4818
BJ [31]	0.8196	0.8416	0.6993
GSC [74]	0.8253	0.8463	0.6977
GSCSEQ	0.8290	0.8482	0.7014
RW [73]	0.7953	0.7440	0.6207
SP [96]	0.7712	0.8092	0.6272
MARG [148]	0.7907	0.8110	0.6354
FCN8s	0.9227	0.8978	0.8573
LFCN8s	0.9549	0.9316	0.8837
FCDenseNet103	0.9648	0.9521	0.9077

Table 3.2: Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on Weizmann dataset [13].

	Pixel Acc.	Mean Acc.	Mean IoU
GM [20]	0.7958	0.7993	0.6602
GrowCut [199]	0.7924	0.7982	0.6465
GrabCut [169]	0.7908	0.7953	0.6513
BJ [31]	0.9117	0.9119	0.8569
GSC [74]	0.9076	0.9056	0.8476
GSCSEQ	0.9092	0.9066	0.8492
RW [73]	0.8568	0.8599	0.7561
SP [96]	0.8390	0.8466	0.7258
MARG [148]	0.9131	0.9152	0.8434
FCN8s	0.9193	0.9068	0.8438
LFCN8s	0.9320	0.9227	0.8656
FCDenseNet103	0.9486	0.9391	0.8847

Table 3.3: Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on Alpha Matting dataset [165].

	Pixel Acc.	Mean Acc.	Mean IoU
GM [20]	0.8313	0.7731	0.6226
GrowCut [199]	0.6600	0.6968	0.4425
GrabCut [169]	0.6571	0.6922	0.4391
BJ [31]	0.8108	0.8069	0.6406
GSC [74]	0.8170	0.8087	0.6458
GSCSEQ	0.8193	0.8092	0.6486
RW [73]	0.7787	0.7665	0.5810
SP [96]	0.7914	0.8030	0.6053
MARG [148]	0.7651	0.7871	0.5685
FCN8s	0.9526	0.9155	0.8710
LFCN8s	0.9689	0.9332	0.8927
FCDenseNet103	0.9751	0.9507	0.9147

Table 3.4: Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on MOS dataset [183].

	Pixel Acc.	Mean Acc.	Mean IoU
GM [20]	0.8165	0.7283	0.5787
GrowCut [199]	0.6268	0.6366	0.3999
GrabCut [169]	0.6282	0.6412	0.4028
BJ [31]	0.7559	0.7824	0.5794
GSC [74]	0.7707	0.7879	0.5903
GSCSEQ	0.7724	0.7883	0.5912
RW [73]	0.7014	0.7102	0.5095
SP [96]	0.7602	0.7809	0.5618
MARG [148]	0.7118	0.7218	0.5050
FCN8s	0.9527	0.9201	0.8723
LFCN8s	0.9630	0.9260	0.8801
FCDenseNet103	0.9825	0.9433	0.9295

Table 3.5: Segmentation accuracy comparison between different types of interactive segmentation techniques and our three proposed deep interactive segmentation modules on VOC validation dataset [62].

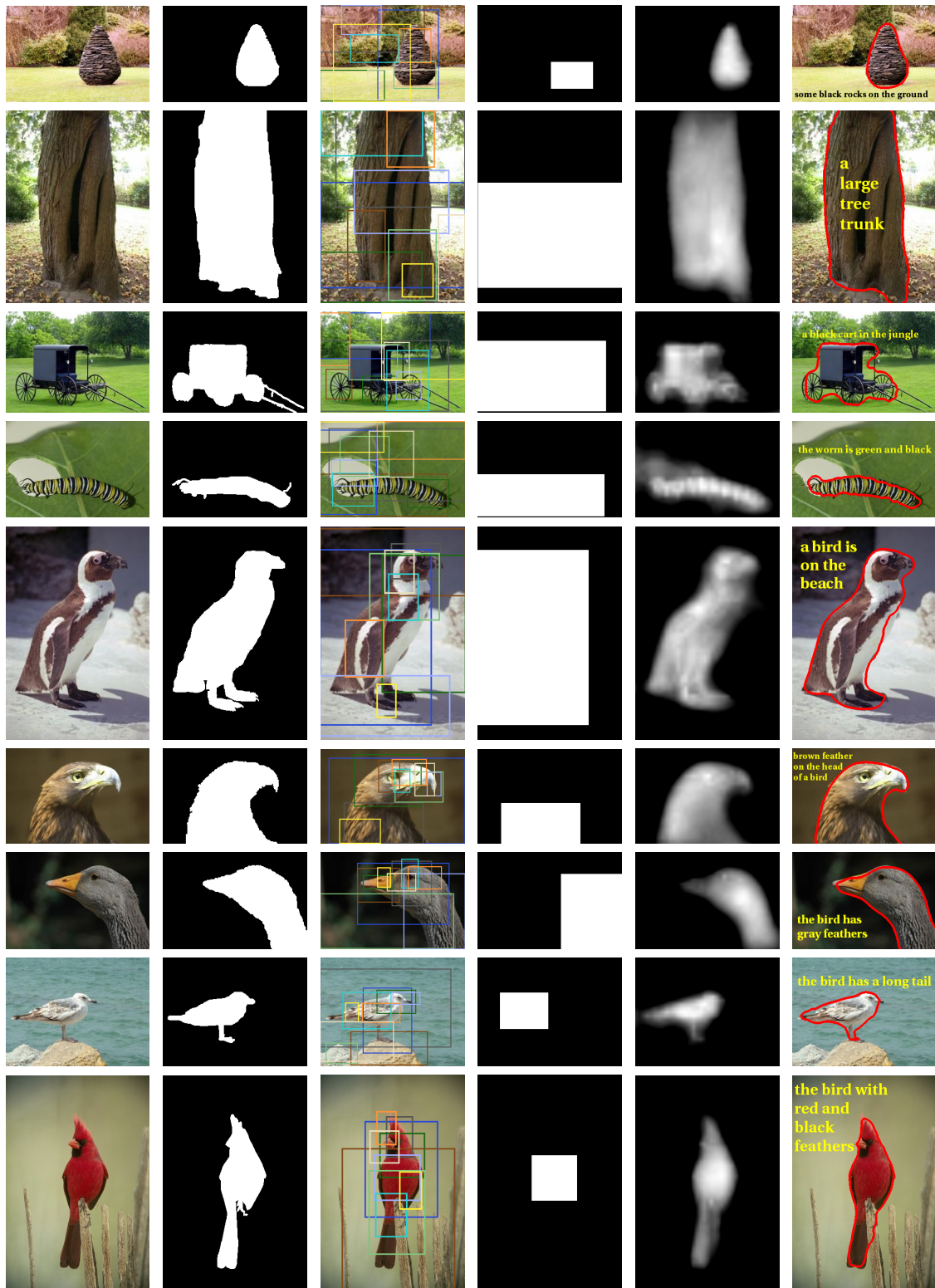


Figure 3-11: Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.

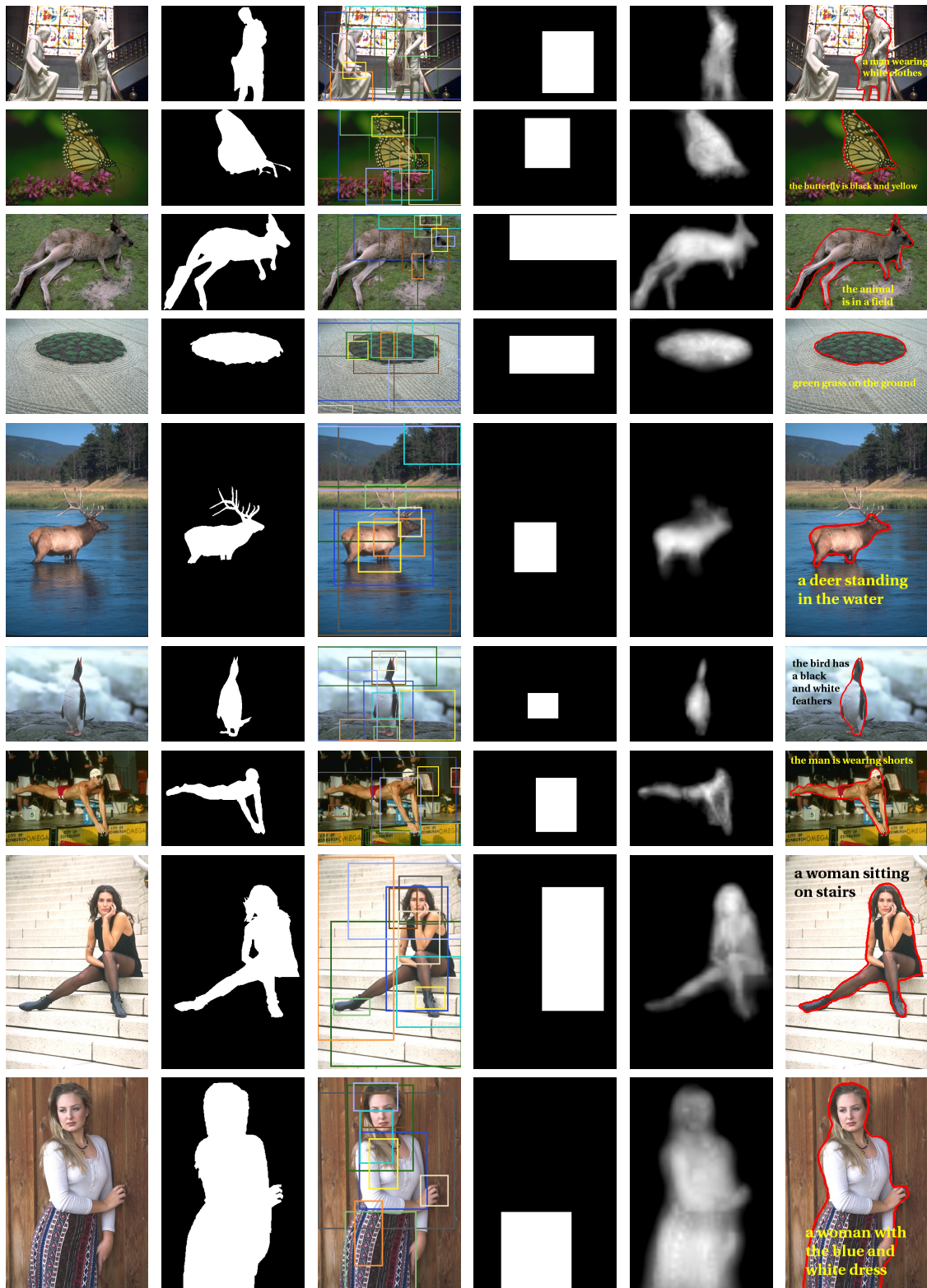


Figure 3-12: Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.

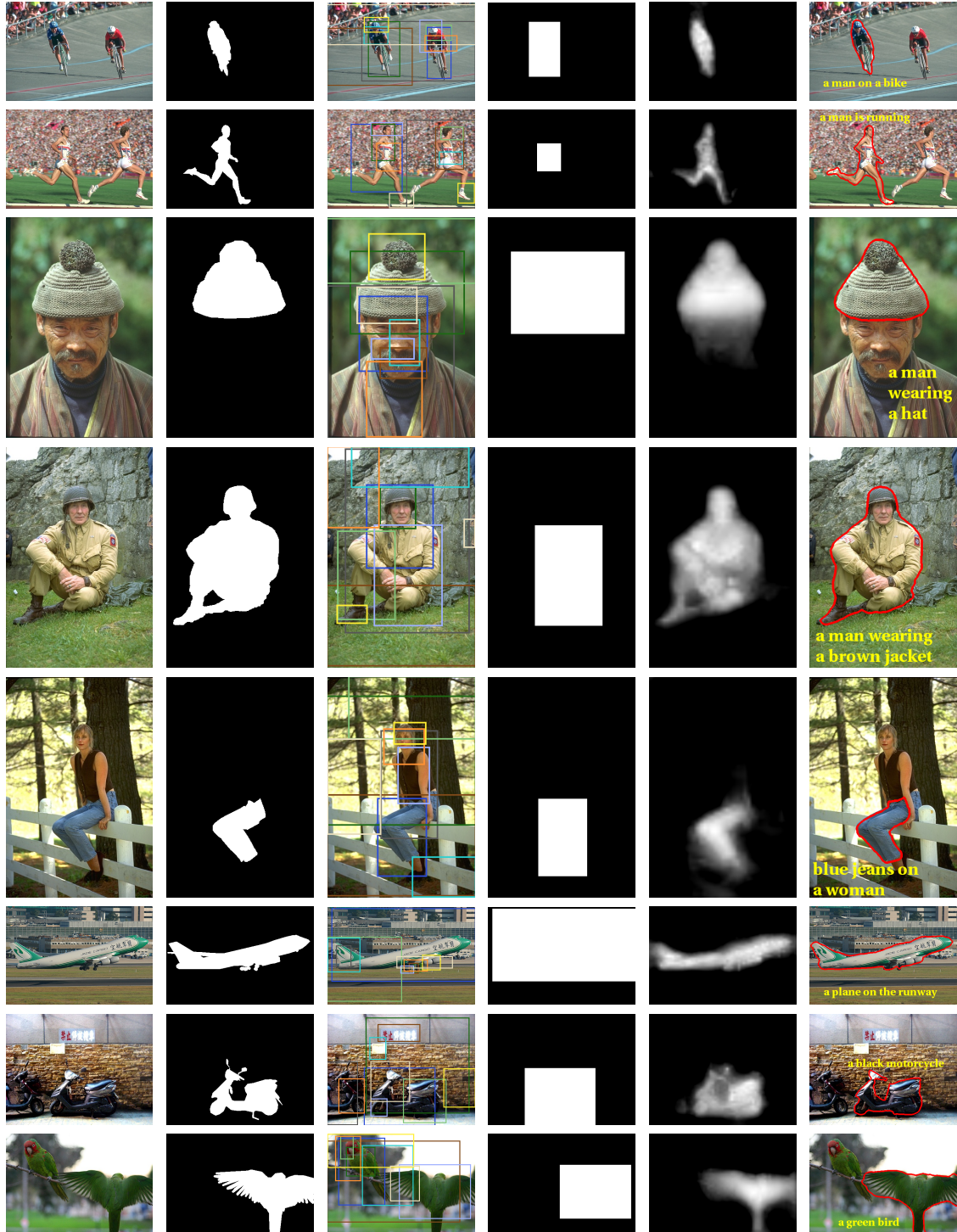


Figure 3-13: Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.



Figure 3-14: Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.

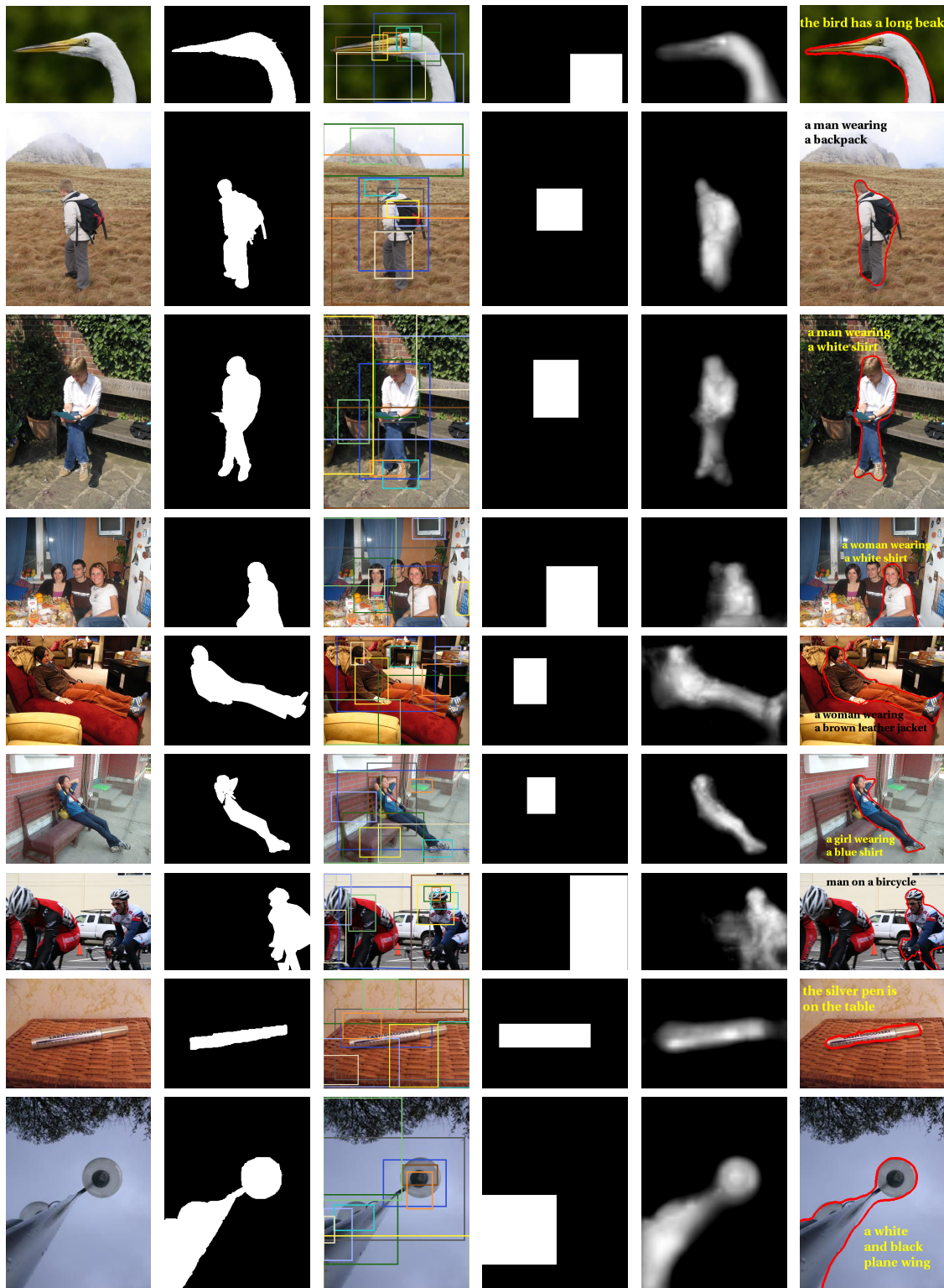


Figure 3-15: Operating progress of our proposed hybrid model, from left to right: input images obtained from [165], [136], [13], [183] or [62], UIR ground truths, dense captioning bounding boxes, best match bounding boxes, the probability maps of our interactive segmentation module and the final outputs of our model including highlighted UIR and its description.

Chapter 4

Deep Selective Texture Labeling

The ultimate goal of this chapter is to establish a *bidirectional correlation* between visual textures and their natural language descriptions. Such a novel technology is able to improve computer vision by providing texture representations that are robust to realistic imaging conditions of the natural scenes. It also boosts content retrieval by providing innovative ways to search and retrieve textures by written explanations. Moreover, it promotes image manipulation by providing new techniques to create and modify textures using descriptions. The main technical contributions of this research is the introduction of:

- *A systematic architecture that combines different aspects of texture modeling with deep learning to enable end-to-end learning of texture representations.*
- *A new model for texture captioning that provides an unprecedented opportunity to describe and also retrieve various texture attributes using natural language descriptions.*

The proposed model has various applications ranging from robotics where understanding material properties of surfaces is an important clue for interaction, to analysis of different forms of imagery.

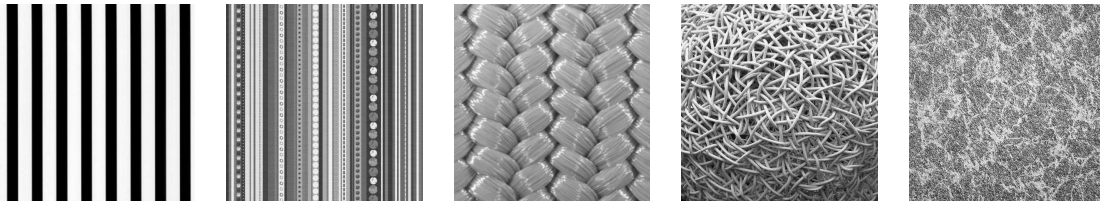


Figure 4-1: Each texture pattern can be located somewhere between regularity (left) and randomness (right). Images are obtained from DTD [43] and ALOT [37] datasets.

4.1 Introduction

Shapes and materials are two main cognitive categories in the visual information. Among several meanings of the word *texture* which are related to different human senses, *tactile texture* can be defined as the set of surface properties that can be felt by touching the surface. In the same way, *visual texture* refer to the visual impression that the surface material produces to the human observer. As illustrated in Fig. 4-1, the pattern of each visual texture can be located on an interval between the perfect *regularity* and the pure *randomness*. Since the surface of any visible object is textured at a certain scale and the texture is the most informative cue to identify different attributes of the homogeneous regions, visual texture is of crucial importance to understand scenes and recognize different regions in digital image processing.

Texture classification, segmentation, synthesis and shape from texture are the most investigated topics in texture analysis. *Texture classification* [150, 151] utilizes a supervised approach to assign each uniform textured region to the texture class it belongs to. *Texture segmentation* [173, 126] aims to identify the visual boundaries of multiple homogeneous texture regions. *Texture synthesis* [116, 76] is the process of producing texture images from detected texture primitives where its most popular application is the rendering of the realistic object surfaces in computer graphics. And finally, *shape from texture* [125, 129] is a 3D reconstruction approach that belongs to a broader class of vision problems known as *shape from X*. In this technique, the ultimate objective is to extract the 3D surface information from a 2D image by estimating the shape of the observed surface from the distortion of the texture created by the imaging process.

4.1.1 Classic Texture Representation

Similar to many other image processing tasks, in texture analysis, the raw pixel values are not informative enough to support an efficient operation. Hence, the extraction of *textural features* plays a significant role to provide a proper *texture representation* for further processing. The traditional texture analysis approaches such as statistical, geometrical, model-based and signal processing methods are widely used to produce useful dictionaries of texture features.

In *statistical methods* [77, 188], each texture is characterized by the distribution of the gray values over the specified region. In this way, *first-order statistics* measure the likelihood of observing a gray value at a randomly-chosen location of the image. Unfortunately, features that are generated by this approach are not able to provide considerable information about the relative positions of the various gray levels within the image. Therefore, *second-order statistics* are defined as the likelihood of observing a pair of gray values occurring at the endpoints of a distance with a random length, location and orientation. In addition, to understand the repetitive nature of the texture elements, *auto-correlation features* are often proposed to assess the regularities as well as the granularity level of the texture [121].

In *geometrical methods* [194, 9], each texture region is supposed to be composed of primitive units called *texture elements* or *textons*. Here, statistical properties and placement rules of these building blocks are used to represent different types of textures. Consequently, such an approach consists of two main steps: texton recognition and texton arrangement analysis. Texton recognition is normally performed by general image processing operations such as edge detection and mathematical morphology, while the placement rules can be analyzed by previously mentioned statistical approaches to investigate the spatial relationships between detected elements.

Model-based approaches [159, 39] aim to find a parametric model to not only capture the essential properties of the texture but also perform texture reproduction process that is known as *texture synthesis*. Markov Random Fields [50] are the most popular tools to model local contextual information. In such a model, the main assumption for the texture analysis is that a pixel intensity can only affect its neighboring area and pixel values are not

dependent at long distances.

Finally, in *signal processing solutions* of the texture analysis [112, 68], certain features are computed from filtered images mostly in the frequency domain. Gabor filters [45] and wavelets [128] as well as quadrature mirror filters [162] appear to be the most popular tools for the frequency decomposition of texture. In these approaches, extracted features are mainly based on statistics of filter responses. It is worthwhile noting that these techniques are based on the manual design of desired features which can be technically complicated and time-consuming. Thus, the performance of the traditional approaches highly depends on the quality of these manually engineered features. Moreover, in most of the cases we encounter multi-scale sophisticated patterns in textures, so typically feature-based dictionaries of these methods cannot properly interpret all observed complexities of the texture. Texture features are expected to extract meaningful and non-redundant information from a large number of pixels. *Local descriptors* provide a robust texture analysis by describing the pixel intensity values within local neighborhoods. Therefore, many successful texture analysis approaches apply *Local Binary Patterns* (LBP) [79, 150] that compute the occurrence histogram of local descriptors and extract local or global features. As a side note, an *occurrence histogram* is an order-less pooling mechanism that summarizes the occurrence of the local descriptors at every pixel location, regardless of their spatial location and neglects the global shape information.

4.1.2 Vocabulary Learning

Vocabulary learning is a set of data-driven approaches that are used to perform an automatic feature extraction and can be considered as the natural extension of classic local descriptors [43]. So, vocabulary learning is a proper framework to generate efficient texture representations. Such an approach is typically consists of a four-step processing pipeline:

- The first step is a sparse spatial sampling of local neighborhoods from which robust and stable descriptors are extracted to describe an image. The basic idea of sparse sampling is to find key points in the image that represent localized patterns. Some robust and fast key point detection methods found in texture analysis are Laplacian

of Gaussian (LoG) or Difference of Gaussian (DoG) filters [194] and Features from Accelerated Segment Test (FAST) [168].

- The next step is the extraction of classic local descriptors such as Histogram of Gradients (HOGs) [52], Scale-Invariant Feature Transforms (SIFTs) [133], Speeded Up Robust Features (SURFs) [23], and LBPs for each key point.
- In the next stage known as *dictionary 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) [164]. Each cluster center is then considered as a *word* in the resulting visual dictionary. Since K-means clustering is not able to capture overlapping distributions in the feature space (as it considers only distances to cluster centers), some other approaches such as Fisher kernel representation also known as *Fisher Vector* (FV) [158] employ *soft clustering* in which clusters are computed by fitting a GMM to the distribution of descriptors.
- In the final step, a *pooling mechanism* is applied to generate final image representation. As mentioned before, the typical pooling method is the histogram of occurrences of visual words from the learned dictionary [117]. The occurrence is often based on the nearest cluster center in the feature space of each local descriptor. Such a pooling technique results in a feature vector which summarizes the occurrence of visual patterns. In *Vector of Locally Aggregated Descriptors* (VLAD) encoding [92], simple occurrences are replaced by distances of local descriptors to the representative visual words. In FV encoding, each Fisher vector is 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 with respect to the mixture weights, means, and covariance matrices. After encoding process, the normalization of the obtained vectors is suggested to improve the matching results. The aforementioned pooling mechanism are order-less, in the sense that the spatial position of local descriptors is discarded and they do not carry the global shape information. Hence, they are useful for texture analysis.

4.2 Deep Texture Representation

With the fast evolution of CNNs for automatic generation of comprehensive feature hierarchies, these architectures are applied as the standard feature extractors. As mentioned before, a proper road map for texture analysis is to design an order-less measure that is able to capture spatial dependencies. For example, computation of features' distribution over image patches provides a proper order-less encoding for texture recognition. Although CNNs did not include such a property in their preliminary design, some recent architectural modifications have tried to embed the new context of deep texture representation in CNNs. In spite of early discussion about incompatibility between the capacity of CNNs and the complexity of texture datasets [21], novel adjustments of CNNs for texture analysis have provided satisfactory improvements not only on texture related tasks but also on other recognition platforms [124, 210].

In the primary attempt of applying CNNs for texture recognition [192], a naive architecture including two hidden layers received the input as a 2D array and produces an output that indicates the texture class of the center pixel. The proposed network applied a different type of neurons called *shunting inhibitory neurons* [27] rather than Sigmoid ones for feature extraction.

In more recent cases of such an application, different mechanisms have been proposed to discard the overall shape information by embedding an order-less pooling layer over extracted feature hierarchies. The main objective for such an order-less encoding approach is to remodel the combination of the detected features in a way that is appropriate for the texture analysis.

In FV-CNN architecture [44], authors transferred the object recognition knowledge of a pre-trained CNN on the ImageNet dataset [172] to the texture feature space where texture descriptors are generated by order-less FV pooling [158] on convolutional filter banks with 64 GMM components. Since the direct finetuning of the Fisher vectors is not trivial, the CNN architecture is only used for feature extraction and its convolutional features are not finetuned by texture samples. Instead of direct finetuning of the whole structure, it is possible to finetune the convolutional layers on the target domain and exploit their modified

parameters to improve the encoding results. Fortunately, such a mechanism can be efficiently used in a region-based approach as it requires computing the convolution output once and pooling the desired regions by its order-less encoding.

In [15], a dedicated CNN architecture (T-CNN) is proposed in which an energy layer is located between convolutional and FC layers to extract the response of the intermediate filters and form a texture representation. The proposed modification facilitates the texture classification, while reducing the number of network parameters by assigning smaller encoding to the final FC layers.

As proposed in [124], various order-less texture descriptors can be written as *bilinear models*. So, they introduced a representation approach that consists of two CNNs as feature extractors whose final feature maps are multiplied using outer product at each location of the image and pooled to obtain an image descriptor. This architecture is able to model the pair-wise feature interactions in a transitionally invariant manner that generalizes order-less texture descriptors. Unlike previous cases, such a model can be easily trained end-to-end that provides a full package of deep learning solution for texture analysis. Since the model is linear in the outputs of two CNNs, it is called *bilinear CNNs*.

As another approach, Deep Texture Encoding Network (Deep-TEN) [210] introduces a novel encoding mechanism for integrating the entire dictionary learning and encoding pipeline into a residual encoding layer. Through the encoding layer, residual vectors are calculated on the pairwise difference between the input visual descriptors and the key points of a learned internal dictionary. Then, the residual vectors are aggregated by the weights that are calculated via soft assignment. The final output of the model is an order-less representation that is particularly useful for material and texture recognition.

4.3 Our Approach for Selective Texture Labeling

Texture is an inherent property of entities, which is related to some surface characteristics such as material, color, pattern, etc. In some cases, texture can be described as the repetition of an element or the appearance of a specific template over a surface. Nevertheless, even for a human, it is not an easy task to find a few illustrative words and describe a texture

in a phrase. So, in spite of this reality that the surface of many objects is textured and visual texture is of key importance to recognize objects and their properties, training samples of existing image captioning datasets that are produced by crowdsourcing e.g. [106, 122] suffer from the lack of linguistic explanations of texture features.

This work is the first attempt to provide a model that makes it possible to automatically generate semantic descriptions for a wide range of textures. Such a conceptual expansion of the labeled visual information provides an unprecedented insight to image captioning techniques and presents an excellent opportunity for their extension into *fine-grained captioning* applicable for the *big data analysis* in geology, meteorology, climatology, oceanography, pedology, agronomy. In addition, introduction of an intelligent agent that is able to understand and describe texture features through natural language conversations can be considered as a significant progress in the design of *recommender systems* that are able to interact with the user to recommend products that satisfy customer's criteria for the color, material and the intrinsic pattern of the desired product.

Based on our previous model proposed in the last chapter and [30], a FCN module is capable of representing not only the visual content of the input image, but also the user intention in a feature hierarchy. As illustrated in Fig. 4-2, the proposed model for the Selective Texture Labeling (STL) exploits a symmetric combination of two FCN modules called *spectators* to encode the visual contents of the input image and the positive and negative clicks of the user that are used to indicate the region that should be explained. Since the occurrence position of the extracted features are encoded in the final feature map of the spectators, their final output is an order sensitive feature hierarchy.

As stated in the previous section, a more useful texture representation is the one that relies on the statistical information of the extracted features rather than their position. So, we need to convert the order sensitive output of the spectators into an order-less representation which can be efficiently used for the texture analysis. Recently, it is shown that order-less representation of texture descriptors can be considered as a bilinear model [124, 123]. Accordingly, the bilinear combination of the spectators' feature maps resembles a quadratic kernel expansion [177] and is able to calculate all the pairwise interaction of the extracted features in an order-less manner. After the merging phase, the resulting set of bilinear fea-

tures are aggregated across the image positions to form a vector representation. Then a normalization step enhances the discrimination strength of the vector representation which is then used as the input of the semantic label generation process.

4.3.1 Connectivity Plan of the Spectator Modules

In traditional convolutional networks, there is only one connection between each layer and its subsequent layer. Recent studies [81, 190] have shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output.

As illustrated in Fig. 4-2, the fully convolutional structure of each spectator module consists of a light-weight *DenseBlock* proposed in the DenseNet [88] architecture where, in each layer, the feature maps of all the previous layers are used as input, and the layer feature map is the input of all the following layers. Hence, a *DenseBlock* of N layers has $N(N - 1)/2$ direct connections. Note that, within the *DenseBlock* all the layers have the same kernel size.

Such a dense connectivity alleviates the vanishing-gradient problem, amplifies feature propagation, encourages feature reuse, and substantially reduces the number of parameters. For each layer, it also provides a direct access to a wide range of features in different levels that makes it possible to generate more compact representation of the data. Moreover, a densely connected architecture requires fewer parameters than traditional convolutional networks, as there is no need to relearn redundant feature maps. In addition, the direct access of layers to gradients from the loss function and the original input signal gives a more smooth decision boundaries and high generation performance to the final classifier that is especially crucial when the amount of training data is insufficient.

Since the concatenation of the feature maps tends to increase the output size of the final layers, there is an upper bound to the number of feature maps that each layer is allowed to produce and append to the main data stream that is known as the *growth rate*. Having g as the growth rate, the n^{th} layer of the proposed spectator module has $5 + g \times (n - 1)$ input size, where 5 is the number RGB channels plus positive and negative user interaction

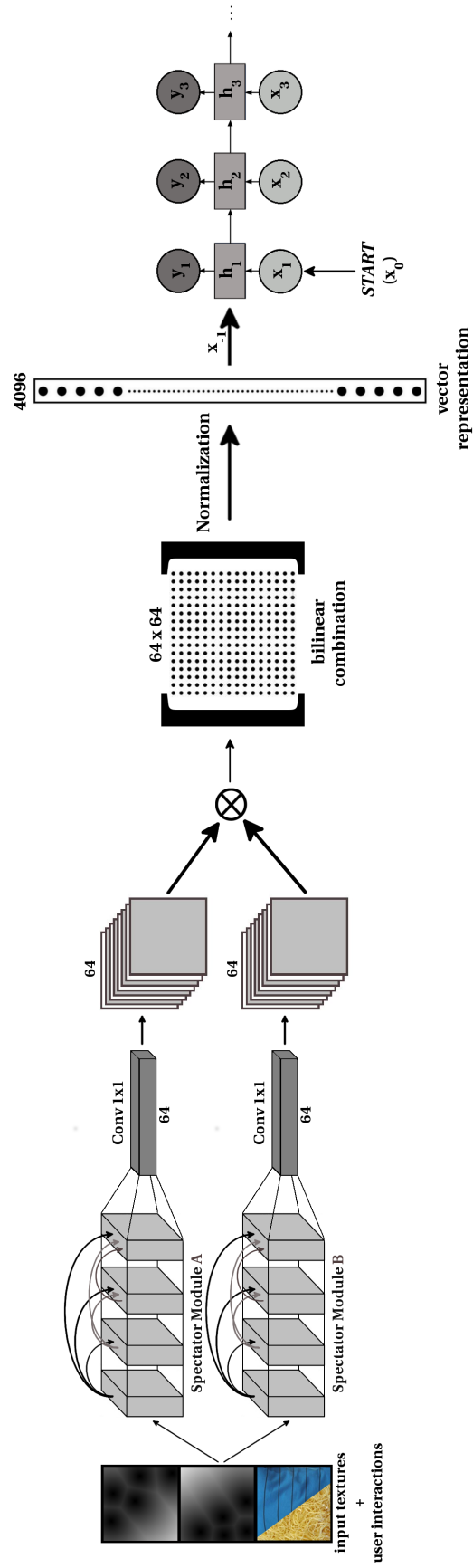


Figure 4-2: Architecture of the proposed deep selective texture labeling model [28].

maps.

4.3.2 Bilinear Texture Representation

The proposed model employs the bilinear pooling mechanism [124, 123] to aggregate the location-wise outer-product of two spectator modules by global averaging. As we applied the symmetric arrangement of the spectator modules (our two spectator modules are identical) to extract a set of features x_i across locations $i = 1, 2, \dots, n$, the outcome of this operation is a *symmetric positive semi-definite* covariance matrix that includes the pair-wise interactions between the extracted features. Thus, the bilinear pooling mechanism is able to extract the second-order statistics:

$$A = \frac{1}{n} \left(\sum_{i=1}^n x_i x_i^T \right) \quad (4.1)$$

where, each feature x_i is a *d-dimension* vector and the matrix A is of size $d \times d$.

Since the spectator modules have the FCN architecture, they are able to accept input image at any size and produce its corresponding feature map. So, after bilinear combination, we are left with bilinear representation of varying sizes. In order to feed the RNN language model (see section 4.3.3), we must provide a fixed-size representation for each texture. To tackle this challenge, after each spectator module, we put a 1×1 convolutional layer with 64 kernels to fix the number of channel in the final feature map. So, the d parameter (dimension of feature vectors) is always fixed to 64 and consequently, the bilinear combination outcome is a fixed-size matrix of size $64 \times 64 = 4096$, no matter what the input size is.

For the sake of performance improvement, the resulting matrix A can be normalized by element-wise signed square root and L_2 normalization [124]:

$$\tilde{A} = \text{sign}(A) \sqrt{\frac{A}{\|A\|_2}} \quad (4.2)$$

After normalization, the resulting matrix will be flattened to form the fixed-size input vector of the next stage.

4.3.3 Semantic Label Generation

Primary techniques that address the problem of generating image descriptions relied on sentence templates, which confined their variety. The main reason for such a restriction was to reduce the complex visual scenes into a single sentence.

With the development of the large image captioning datasets [106, 122], image captioning approaches [30, 100, 95, 200, 202, 134, 206] relaxed this restriction and relied on direct learning from the data where the model induced the latent relation between parts of the sentences and regions of the image that they describe. The most common architecture for such a model is the *RNN-based language model* [141, 55] that takes an input image and generates its linguistic description in text. RNNs are very useful to capture long-term dependencies and therefore seem ideal to describe semantic relations in the form of sentences and even paragraphs [105].

The visual content of a texture region does not include semantic relations between its internal elements and therefore can be described by a fixed-size sentence wherein each word represents one of its general attributes such as surface entity, pattern, material, and color.

The central element of the semantic label generation process is a multi-modal RNN language model which is conditioned on bilinear texture representation. During the training phase of the proposed model, the language model receives the normalized bilinear representation of the input texture followed by a sequence of tokens (x_0, x_1, \dots, x_n) where x_0 is the special START token and (x_1, \dots, x_n) represent the texture attributes of the training sample. Here, each description is a sequence of attributes drawn from a pre-determined vocabulary V that includes *one-hot* vectors [142] of all the existing texture attributes in the training data (see section 3.3.3).

The recurrent language model generates sequences of hidden states $(h_1 \dots h_T)$ and output vectors $(y_1 \dots y_T)$. The output vector y_t is of size $|V + 1|$ and holds the unnormalized log probabilities of the attributes in the texture vocabulary and the additional special token END that is expected to be produced at the end of each generated description. The hidden states are initialize by a zero and the bilinear representation of the input texture is presented to the model once at the beginning.

At the first time step of each training iteration, the expected outcome for y_1 is the first attribute in the ground truth texture description. At the next time step, the network input x_2 is the *one-hot* vector of the first attribute in the ground truth description and expected output y_2 is the second attribute of the given description. We iterate this process until the time step in which the network generates the special END token.

4.4 Experiments

During training, the input to our model is a set of texture images that are equipped with various texture attributes as descriptions. These training samples are the result of a pre-processing step that combines two texture datasets to form a multi-label texture dataset. In these texture datasets, categorical labels contain useful explanations of the texture features.

4.4.1 Multi-Label Texture Dataset

In contrast with image captioning datasets that include dense annotations about objects, their attributes, and the observable relationships between visual components, texture datasets [25, 182, 113, 111, 78, 53, 204, 149] contain no description and their annotations are limited to their labels.

To generate a multi-label dataset of texture images, we combined two well-known texture datasets, each of which has a unique set of labels. The first collection is the *Describable Textures Dataset* (DTD) [43] that consists of 5640 images where each sample has a size between 300×300 and 640×640 pixels. Inspired by human perception, DTD instances are organized according to 47 terms such as “*blotchy*”, “*cracked*”, “*dotted*” and “*honey-combed*” that describe visual patterns. There are 120 images in each category and each image contains at least 90% of the surface representing the category attribute. For each image a key attribute and a list of joint attributes are provided.

The second dataset is the *Amsterdam Library of Textures* (ALOT) [37] that includes 250 texture classes with 100 samples of size 384×256 per class. A wide variety of materials such as “*tobacco*”, “*bread*”, “*sand*”, “*sugar*”, “*cotton*”, “*beans*” and much more are



Figure 4-3: Some samples of ALOT dataset [37] (back) including **material labels** such as *sunflower seeds*, *sheep wool* and *spaghetti*, and DTD dataset [43] (front) providing **pattern labels** such as *banded*, *crystalline*, and *marbled* that are used with **color information** to shape our *multi-label texture dataset*.

presented in this dataset where the samples are captured with eight different illuminations from three orientations and four viewpoints. Some samples of both datasets are shown in Fig. 4-3.

While samples of the DTD dataset represent different human-defined patterns and the ALOT dataset covers a wide range of materials, the color property of the textures is not appeared in their labels. As illustrated in Fig. 4-4, we utilized the *grid sampling* and the *average pooling* strategy to extract the dominant color of each training samples and equip its linguistic description with such an important visual attribute. In each image, we sampled 1000 pixels from each channel to compute RGB values of the dominant color. Finally, the resulting values will be converted into the nearest color name based on *CSSW3C* standard [38].

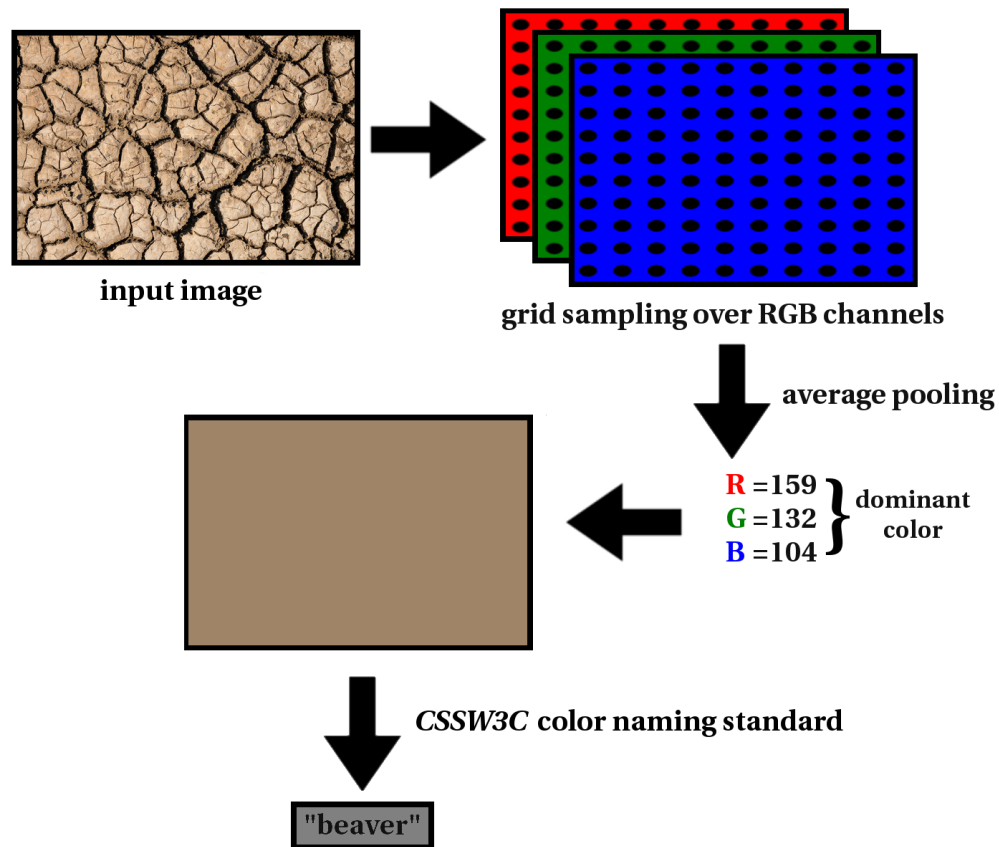


Figure 4-4: Extraction of the dominant color's name via grid sampling and average pooling based on *CSSW3C* standard [28].

In pictures containing natural landscapes, there are several textured regions with different scales that can be determined by user clicks to be described. Unfortunately, there are several obstacles to develop a combination of a dataset and a deep architecture that performs such a task properly. First, small textured regions do not contain enough information for a robust extraction of the tiny texture attributes. In addition, uncontrolled illumination conditions of the outdoor scenes have a destructive influence on the grasp of the texture properties. Furthermore, as mentioned before, even humans are not able to provide illustrative explanations for texture attributes which is the main barrier to provide a texture captioning dataset. Last but not least, texture captioning has specific applications on the pictures that are captured under specific conditions such as quality control, advertisement and satellite imagery. So, the generation of a dataset that is able to simulate these conditions seems to be satisfying.

Normally, before the training process of the deep architectures, rotation, scaling and translation are used in a so called *data augmentation* preprocessing step to increase the number of training samples. Since the appearance of the texture is invariant to these transformations, embedding different textures in each training sample is one of a few tricks that provides an opportunity to generate more training samples. In addition, having combinatorial samples lets us exploit the interactive property of the spectator modules that are pretrained among the model of the previous chapter to add an interactive functionality to our texture labeling module.

To this aim, we construct each training sample by a combination of two distinct textures of size 256×256 in *vertical*, *horizontal*, *diagonal*, *square segment* and *circle segment* arrangements as shown in Fig. 4-5. For each training sample, all arrangements are generated by random selection of offset values. In each arrangement, the black area of the binary mask indicates the positive texture that is described by the sample description and the white area is considered as the negative texture which is not explained. During the generation process of each combination, positive and negative textures are switched by the probability of 50%.

The model receives five random clicks inside each region to determine positive and negative textures. Two sets of positive and negative clicks are then used separately to generate two Voronoi diagrams that are feed into spectator modules as the user interactions (see Fig.

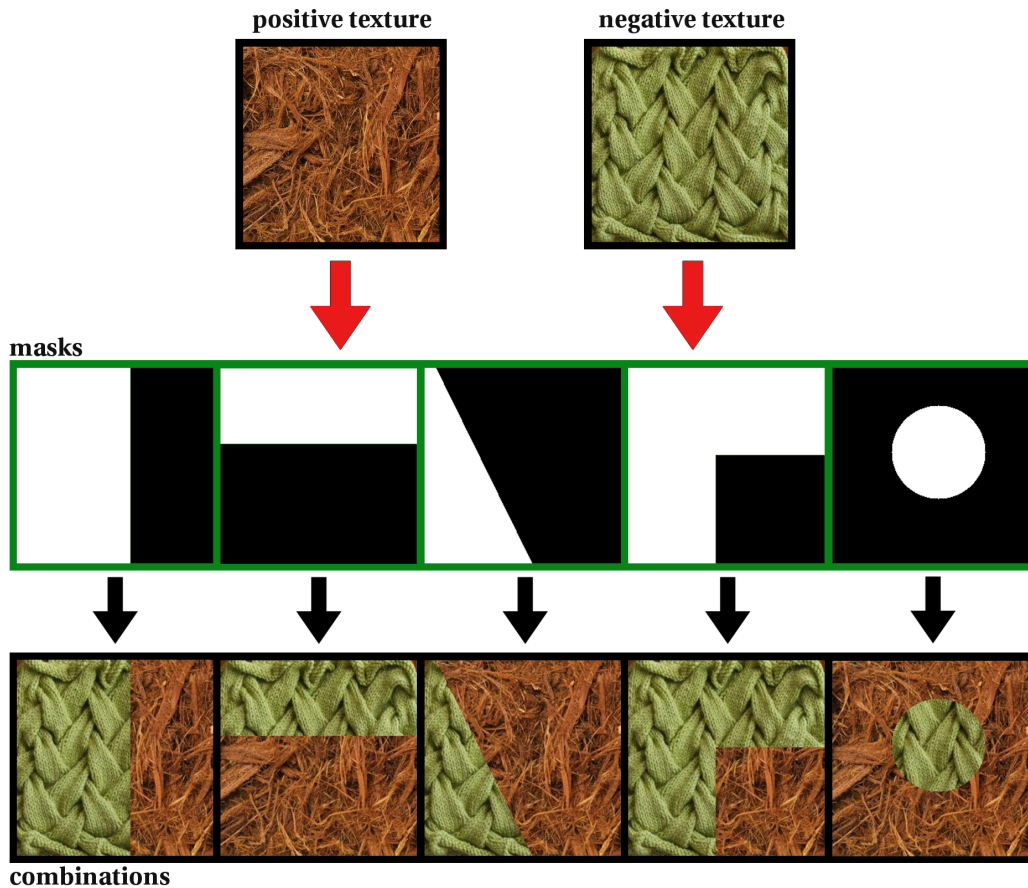


Figure 4-5: Each training sample includes a positive and a negative texture that are combined based on one of different arrangements determined by randomly generated binary masks (black is positive) [28].

3-3 and section 3.4.1).

We randomly chose ten instances per class to combine two datasets which leads to 1,175,000 combinations that in half of them ALOT textures are considered positive. At the next step of the data preparation, we dropped combinations in which the positive texture description has appeared less than 100 times in the whole dataset (mostly rare colors). Doing so, we are left with 874,320 combinations. Finally, we selected 500,000 combinations that possess longer descriptions. We used 75% of the data (400,000 samples) for training and preserved the rest for the model evaluation.

4.4.2 Training and Optimization

We trained the full model end-to-end in a single step of optimization. We initialized the spectator modules with weights pre-trained on the task of previous chapter (first *Dense-Block* of the *FCDenseNet103* interactive segmentation module) and all other weights with a Gaussian with standard deviation of 0.01. We use stochastic gradient descent with momentum 0.9 to train the weights of the spectator modules. We also used the AdaDelta [208] to train the RNN language model with the following updating formula:

$$(v_t)_i = \frac{RMS((v_{t-1})_i)}{RMS(\nabla L(W_t))_i} (\nabla L(W_t))_i \quad (4.3)$$

where

$$RMS(\nabla L(W_t))_i = \sqrt{E[g^2] + \epsilon}, \quad (4.4)$$

$$E[g^2]_t = \delta E[g^2]_{t-1} + (1 - \delta)g^2_t \quad (4.5)$$

and

$$(W_{t+1})_i = (W_t)_i - \alpha(v_t)_i. \quad (4.6)$$

Here for each network parameter i , RMS is the root mean square function and v_t is the weight update value. Moreover, $\nabla L(W_t)$ is the weights' gradient, $E[g^2]_t$ is the decaying average of the squared gradients, W_t is the current weights matrix and W_{t+1} is the updated weights matrix. Finally, the learning rate α is 0.1 and the weight decay δ is $5e - 4$. Our training batches consist of 16 training samples. Since the initial value of the parameters acts as a regularizer of the optimization process, we begin fine-tuning the layers of the spectator modules after one epoch.

4.4.3 Qualitative Results

In this section, we show example predictions of the deep selective texture labeling model. Fig. 4-6 illustrates the output of the proposed model for a number of randomly selected test samples in our multi-label texture dataset. It is shown that, despite the lack of material information in some ground truth descriptions, surface materials such as stone, wood,

cotton, and salt are recognized correctly. In addition, the model has an outstanding generalization strength in the context of color information. Note that, ground truth descriptions of some training samples such as the third one in Fig. 4-6 are not so informative. However, the model succeeded in using its generalization capability to provide more informative descriptions for those instances. In addition, the pattern identification competency of our model is remarkable as it is able to provide such an information for all the test samples which we think is due to extensive inter-class variations of DTD dataset.

Some cases of false label generation are also shown in Fig. 4-7. As it can be seen, in these experiments the model is not able to recognize texture material. These samples are rarely appeared in the training data due to our random selection from DTD and ALOT datasets.

4.4.4 Model Performance Evaluation

During a set of experiments, we evaluated the performance of the proposed model on UIUC [113], Kylberg [111], and KTH-tips-2b [78] datasets. As there is no convention for texture description, different texture datasets utilize various vocabularies that are not necessarily compatible. Moreover, none of these datasets provides more than one label for its instances. To resolve these problems, we replaced the class names of the aforementioned datasets with the most relevant multi-label descriptions that explain the corresponding classes in our multi-label space. Label conversion protocols are demonstrated in Tables 4.1, 4.2, and 4.3. UIUC and Kylberg datasets are comprised of gray-scaled images, so we used the color extraction process of section 4.4.1 to generate names of gray shades for those instances and replaced the [COLOR] tag with those names. For KTH-tips-2b samples, color name extraction process is exactly the same as our proposed approach explained in section 4.4.1. Since *Wall* and *Brick* classes of the UIUC dataset are not appeared in the training data, we excluded them from our evaluation process.

At the next step, we investigated the ability of our proposed model to describe texture attributes. We first trained our model on full image textures (not combinations) with the aim of verifying that the model is rich enough to support the mapping from visual textures to sequence of color, material and pattern attributes without user interactions. We named this

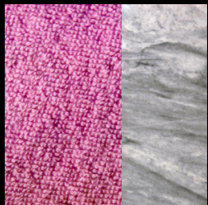


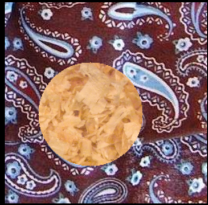
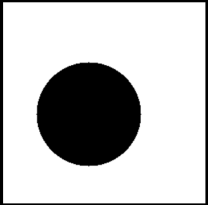








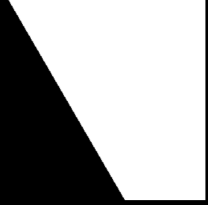


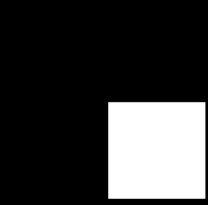



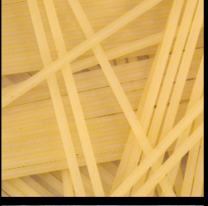

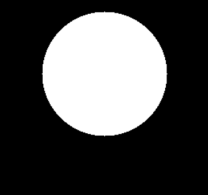

			gray blotchy marble
			gray marbled stone
			dark salmon wood fiber
			fawn fibrous wood fiber
			tan woven
			dark brown woven wood
			orange orange parts
			amber veined orange
			light slate knitted
			cool gray knitted cotton
			earth yellow spaghetti
			chrome yellow crosshatched spaghetti
			lavender gray rock salt
			lavender gray crystalline salt

Figure 4-6: Some qualitative results of the proposed model. From left to right: test samples obtained from [113], [111], or [78], corresponding binary masks (black is positive), positive textures in test samples, ground truth descriptions (in black), and predicted descriptions by the proposed model (in blue).

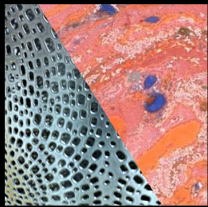
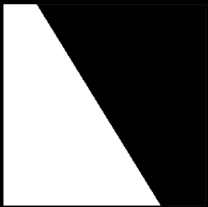
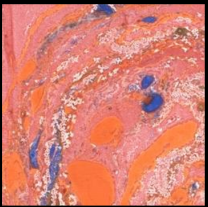
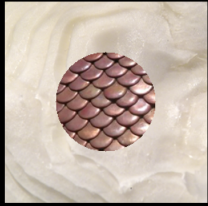
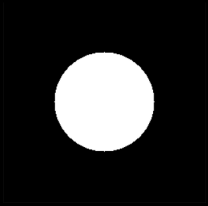


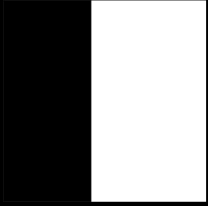

			light coral blotchy marble
			newyork pink chili souce
			white coal
			pastel gray matted wool
			white bread
			pale silver perforated sponge

Figure 4-7: False samples of the proposed model. From left to right: test samples obtained from [113], [111], or [78], corresponding binary masks (black is positive), positive textures in test samples, ground truth descriptions (in black), and predicted descriptions by the proposed model (in red).

setting as the "full-image non-selective texture labeling". Then the model performance is measured during its full functionality in presence of user interactions called "selective texture labeling on dual texture combinations". Tables 4.4 and 4.5 report the average of the normalized *Bilingual Evaluation Understudy* (BLEU) [155], *Consensus-based Image Description Evaluation* (CIDEr) [198], and *Metric for Evaluation of Translation with Explicit ORdering* (METEOR) [54] scores for both experiments on UIUC, Kylberg, and KTH-tips-2b datasets. All three metrics define the similarity over words or *n-grams* of the predicted and the ground truth descriptions by considering different formulas. BLEU is one of the first metrics that have been used for measuring similarity between two sentences. It has been initially proposed for machine translation, and defined as the geometric mean of *n-gram* precision scores multiplied by a brevity penalty for short sentences. In other words, the BLEU measure looks at the presence or absence of particular words. Since in our case, grammar and the word order of the generated description are not important, we used uni-grams (word-by-word comparison) and the results are reported as BLEU₁:

$$\text{BLEU}_1(p, g) = \frac{\sum_{\omega \in p} \min(C_p(\omega), C_g(\omega))}{\sum_{\omega \in p} C_p(\omega)} \quad (4.7)$$

Here, p and g are the predicted and the ground truth descriptions. ω represents uni-grams (words) of the predicted description, while $C_p(\omega)$ and $C_g(\omega)$ stand for the *Count* of the word ω in p and g , respectively.

CIDEr is a more recent metric proposed for evaluating the quality of image descriptions. In our experiment, it measures the consensus between the predicted and the ground truth descriptions. To this aim, all words of the both descriptions are first mapped to their stem. Then, each description is represented by the set of words present in it. Next, the measure of consensus encodes how often the words in the predicted description are present in the ground truth description. Similarly, words that are not presented in the ground truth description should not be in the predicted description. As a side note, words that commonly occurred in the descriptions of texture samples should be given lower weights, since they are likely to be less informative. To encode this intuition, the *Term Frequency Inverse Doc-*

ument Frequency (TF-IDF) weighting is performed for each word ω_k that computes two weights: *TF* that measures the frequency of the word ω_k in description x , and *IDF* that computes ω_k rarity:

$$G_k(x) = \underbrace{\frac{h_k(x)}{\sum_{\omega_l \in \Omega} h_l(x)}}_{TF} \underbrace{\log \left(\frac{|I|}{\sum_{I_p \in I} \min(1, h_k(x_p))} \right)}_{IDF} \quad (4.8)$$

where $h_k(x)$ is the number of times ω_k is appeared in the description x , $h_k(x_p)$ is the number of times ω_k is appeared in the description of the image p , Ω is our vocabulary and I is the set of all images in the data set.

Then the CIDEr score for each pair of predicted and ground truth descriptions (p, g) is computed using the *average cosine similarity* between the predicted description p and the ground truth description g :

$$\text{CIDEr}(p, g) = \frac{G(p) \cdot G(g)}{\|G(p)\| \|G(g)\|} \quad (4.9)$$

where $G(\cdot)$ is a vector formed by $G_k(\cdot)$ corresponding to all words in the description (\cdot) and $\|G(\cdot)\|$ is the magnitude of the vector $G(\cdot)$.

METEOR is another machine translation metric. It is defined as the *harmonic mean* of *precision* and *recall* of matched words between the predicted and the ground truth descriptions. Additionally, it makes use of synonyms and paraphrase matching. The algorithm first creates an *alignment* between predicted and ground truth descriptions as a mapping between words. A mapping can be thought of as a line between a word in one description, and a word in the other one. As a constraint, every word in the predicted description must map to zero or one word in the ground truth description. If there are two alignments with the same number of mappings, the alignment is chosen with the fewest crosses, that is, with fewer intersections of two mappings. Stages are run consecutively and each stage only adds to the alignment those words which have not been matched in previous stages. Once the final alignment is computed, words' *precision* P is calculated as:

$$P = \frac{m}{w_p} \quad (4.10)$$

where m is the number of words in the predicted description that are also found in the ground truth description, and w_p is the number of words in the predicted description.

In addition, words' *recall* R is computed as:

$$R = \frac{m}{w_g} \quad (4.11)$$

where m is as above, and w_g is the number of words in the ground truth description. Precision and recall are combined using *harmonic mean* with recall weighted 9 times more than precision:

$$\text{METEOR}(p, g) = \frac{10PR}{R + 9P} \quad (4.12)$$

This measure accounts for congruity with respect to single words that appear in both the predicted and the ground truth description. Since in our case, we only performed word-by-word comparison, we reported this measure as the final METEOR value.

Our proposed model is quite successful to generate meaningful descriptions similar to ground truth texture attributes. This can be understood by the high values of BLEU-1 and CIDEr metrics in both experiments as the sign of significant overlap between generated and ground truth texture descriptions. Normally, METEOR looks for synonyms to capture overall semantic similarity and provides lower scores.

Moreover, the model performs slightly better when exploiting its entire capacity to represent texture attributes that the situation in which a part of capacity is dedicated to recognize user interactions.

Finally, we guess that the reason of higher scores on UIUC and Kylberg datasets is their low color variation (just shades of gray) which is easier to learn than the wider color diversity of KTH-tips-2b instances.

UIUC	
Original Class Name	Our Alternative Description
bark1	[color] grooved tree bark
bark2	[color] blotchy tree bark
bark3	[color] stripped tree bark
wood1	[color] swirly wood
wood2	[color] stratified wood
wood3	[color] stripped wood
water	[color] wrinkled water
granite	[color] flecked marble
marble	[color] blotchy marble
floor1	[color] flecked carpet
floor2	[color] flecked carpet
pebbles	[color] bumpy gravels
wall	[NOT USED]
brick1	[NOT USED]
brick2	[NOT USED]
glass1	[color] lace-like plastic
glass2	[color] bumpy flecked carpet
carpet1	[color] blotchy flecked carpet
carpet2	[color] bumpy interlaced carpet
upholstery	[color] dotted carpet
wallpaper	[color] bumpy wallpaper
fur	[color] matted fur
knit	[color] woven knit
corduroy	[color] grooved striped cloth
plaid	[color] grid cloth

Table 4.1: Label conversion protocol between our multi-label dataset and UIUC [113] dataset.

Kylberg	
Original Class Name	Our Alternative Description
blanket1	[color] blotchy flecked wool carpet
blanket2	[color] zigzagged blanket
canvas1	[color] bumpy flecked cloth
ceiling1	[color] bumpy flecked carpet
ceiling2	[color] perforated plastic
cushion1	[color] bumpy banded carpet
floor1	[color] lace-like wood
floor2	[color] bumpy interlaced carpet
grass1	[color] fibrous fabric
lentils1	[color] bumpy beans
linseeds1	[color] bumpy linseeds
oatmeal1	[color] rolled oats
pearlsuger1	[color] bumpy pear sugar
rice1	[color] bumpy rice
rice2	[color] bumpy wheat grains
rug1	[color] bumpy interlaced carpet
sand1	[color] flecked sands
scarf1	[color] lace-like cloth
scarf2	[color] banded cloth
screen1	[color] banded crosshatched carpet
seat1	[color] crosshatched cloth
seat2	[color] blotchy flecked cotton carpet
sesameseeds1	[color] bumpy sesame seeds
stone1	[color] flecked marble
stone2	[color] flecked blotchy marble
stone3	[color] flecked marble
stoneslab1	[color] flecked gravels
wall1	[color] flecked bumpy gravels

Table 4.2: Label conversion protocol between our multi-label dataset and Kylberg [111] dataset.

KTH-tips-2b	
Original Class Name	Our Alternative Description
crumpled aluminum foil	[color] wrinkled silver foil
cork	[color] flecked cork
wool	[color] matted [woven] wool
lettuce leaf	[color] veined leaf
corduroy	[color] grooved striped cloth
linen	[color] knitted wool cloth
cotton	[color] knitted cotton cloth
brown bread	[color] porous bread
white bread	[color] porous bread
wood	[color] swirly wood
cracker	[color] bumpy porous cracker

Table 4.3: Label conversion protocol between our multi-label dataset and KTH-tips-2b [78] dataset.

Full-Image Non-Selective Texture Labeling			
Metric	BLUE-1	CIDEr	METEOR
UIUC	0.737	0.778	0.342
Kylberg	0.763	0.819	0.323
KTH-tips-2b	0.667	0.711	0.291

Table 4.4: Diverse captioning scores of our proposed model on three different external datasets for full-image non-selective texture labeling experiments.

Selective Texture Labeling on Dual Texture Combinations			
Metric	BLUE-1	CIDEr	METEOR
UIUC	0.682	0.714	0.315
Kylberg	0.663	0.744	0.280
KTH-tips-2b	0.618	0.691	0.261

Table 4.5: Diverse captioning scores of our proposed model on three different external datasets for selective texture labeling experiments on dual texture combinations.

4.5 Conclusion

In this chapter, for the first time ever, we presented a hybrid CNN-RNN model that combines different aspects of the deep texture representation with the concept of the automatic image captioning and enables end-to-end learning of the selective texture labeling. Our novel architecture provides new opportunities to describe, search, and also retrieve texture images from their linguistic descriptions. To be able to train such a model, we generated a multi-label texture dataset that covers color, material, and pattern labeling simultaneously. This dataset is the result of the first attempt to provide such an information and should be extended to further texture descriptive aspects. Through several experiments, we investigated the performance of our proposed model in selective and non-selective modes to automatically generate semantic descriptions for a wide range of textured surfaces and measured its accuracy on several unseen texture datasets by different captioning evaluation metrics. Our contribution to the automatic generation of texture descriptions provides an excellent opportunity to enrich the existing vocabulary of the image captioning. Such a conceptual extension can be used for fine-grained captioning applicable in geology, meteorology, climatology, oceanography, pedology, agrology, and recommender systems.

Chapter 5

PDE-based Color Morphology using Matrix Fields

The previous chapters of this thesis were focused on deep learning architectures and their application in computer vision. This chapter relates to the work done during my Ph.D. that is not directly related to deep learning. In this chapter, we propose a new approach to perform morphological operations on color images. This work is appeared in proceeding of the fifth international conference on Scale Space and Variational Methods in computer vision (SSVM 2015) [29], in Bordeaux, France. The final publication is also available at link.springer.com. During the proposed approach, we convert *rgb*-values of the pixels into symmetric 2×2 matrices, where the new color space can be interpreted geometrically based on the HCL biconal color space structure.

Motivated by the formulation of the fundamental morphological operations dilation and erosion in terms of partial differential equations (PDEs), we show how to define finite difference schemes making use of the matrix field formulation. The computation of a pseudo supremum and a pseudo infimum of three color matrices is a crucial step for setting up advanced PDE-based methods. We show that this can be achieved by an algebraic technique. We investigate our approach by dedicated experiments and confirm useful properties of the new PDE-based color morphology operations.

5.1 Introduction

With abundant sources of visual color information such as smartphone, tablets and digital cameras, it becomes increasingly important to consider this information in the construction of image processing tools. Following the seminal work of Serra and Matheron [138, 180], morphological operations form a fundamental class of image processing techniques. Morphological processing is a set of nonlinear operations that are applied on pixels arranged in structuring elements. The building blocks of mathematical morphology for gray-scale images are the processes of *dilation* and *erosion*. Almost all the other morphological operations such as opening, closing, hit and miss transform, thinning, thickening and skeletonization can be derived from these two fundamental operations.

Considering the important underlying mathematical structure of these operations, it is required to define a total order of the values that are contributing in the structuring elements. Although for gray-scale images the corresponding lattice theory framework seems to be satisfactory and adequate, the extension of this concept to color spaces is not trivial. In such a situation, the main obstacle is the lack of a total order for vector-valued data such as *rgb* values. Hence, performing even the simplest morphological operation on color images is a great computational challenge.

There have been numerous attempts to establish a proper morphological framework for color images. A vast majority of these attempts are concentrated on the definition of a ranking schemes and appropriate extremal operators as the substitutes for maxima and minima [16, 47, 72, 181]. For a conceptually different development, It is worthwhile to mention that the approach by Van de Gronde *et al.* [196] that relies on a partial order rather than a total order. However, one may conclude that the optimal way to define morphological operations on color images is still an open issue and that a proper solution might depend on the purpose of the filtering.

In this chapter, we tackle the issue by combining two groups of existing approaches to formulate our novel strategy for color image morphology. The first concept we consider is the formulation of dilation and erosion in terms of Partial Differential Equations (PDEs) [14, 18, 33, 176]. Mimicking a special wave propagation process, the arising PDEs are

hyperbolic Hamilton-Jacobi equations. Then, important numerical methods for discretizing the PDEs for dilation and erosion in the gray-value setting are the schemes of Rouy and Tourin [170], Osher and Sethian [154] and the flux-corrected transport (FCT) scheme of Breuß and Weickert [32]. Motivated by these developments and driven by an interest to filter data arising in diffusion tensor magnetic resonance imaging (DT-MRI), the PDE-based approach as well as the above mentioned schemes have been generalized to deal with specific matrix fields, see e.g. [34, 35] and the references therein. The matrices defining the data for these PDE-based morphological methods are symmetric, positive semi-definite and of dimension three times three.

Secondly, we consider the developments in the recent work [36]. There, color images are embedded into matrix fields consisting of symmetric 2×2 matrices. For these, matrix-based operations are described that mimic dilation/erosion in the spirit of the classical, set-theoretic approach.

As indicated we combine in this work the above mentioned developments in defining PDE-based methods for mathematical morphology of color images. We employ the framework presented in [36] to transform *rgb* data into a bicone-shaped color space that corresponds to symmetric 2×2 matrices. For such matrices we define finite difference schemes that describe in the discrete sense the PDEs of morphological dilation/erosion.

While on the technical side this translation of the schemes as described e.g. in [34] to the color matrix framework seems at a first glance to be relatively straightforward, it is beneficial to state some issues. First, the matrices we deal with here are not positive semi-definite. Thus, taking over technical parts from methods developed in the aforementioned DT-MRI context may not lead to useful results. Secondly, and as a technical difference to the proceeding in [36], we do not employ here the procedures of addition and subtraction motivated by *Einstein addition* in Hilbert spaces. Furthermore, and again in the light of the many attempts in previous literature [16, 47, 72, 181], it is not self-evident that one obtains reasonable numerical results when constructing a method for the purpose of color morphology. However, for our approach we confirm experimentally that it does not give so-called false colors, cf. [181]. This means, that our PDE-based dilation and erosion processes may only lead to color modifications in the sense that they appear in higher and lower saturated

versions of contributing colors, and not as a completely different color.

5.2 PDE-based Morphology

In this section, we first give a brief account of the two operations that are at the basis of our developments, namely morphological dilation and erosion. As we seek to emphasize the underlying ideas here, we stick to a simple presentation. A structuring element E is a mask that allows us to specify neighborhood structures in an image. Then one may use SE s to define morphological operators acting on them. For a given, initial image f we write the *dilation* and the *erosion* with such a structuring element E as:

$$f \oplus E := \sup\{f(x - x', y - y') \mid (x', y') \in E\} \quad \text{and} \quad (5.1)$$

$$f \ominus E := \inf\{f(x - x', y - y') \mid (x', y') \in E\} \quad (5.2)$$

respectively. Making use of these building blocks, one can define e.g. morphological derivative operators. One which is useful in the context of this work is the so-called *morphological Laplacian* [197] which reads as:

$$\Delta_E f := (f \oplus E) - 2f + (f \ominus E) \quad (5.3)$$

As it is evident, the morphological *Laplacian* is a morphological counterpart of the second derivative of a function. It allows to distinguish regions influenced by brightness minima and maxima in an image. As mentioned in [153], this is useful for defining so-called *shock filters*. In the gray-value setting, one step of shock filtering applied pixel-wise at an image f may be described as:

$$S_E f := \begin{cases} f \oplus E, & \Delta_E f < 0 \\ f, & \Delta_E f = 0 \\ f \ominus E, & \Delta_E f > 0 \end{cases} \quad (5.4)$$

As can be seen by considering (5.4), shock filtering amounts to applying dilation and erosion in order to enlarge brightness maxima and minima, respectively, while the transition line between these regions is managed by the morphological Laplacian. In a PDE-based setting as already described in [153], the dilation and erosion PDEs are solved iteratively in accordance to the process (5.4).

Thinking of a gray-valued image as a discrete representation of a continuous-scale function, some of the geometric characteristics of continuous morphology are omitted in its discrete version. As an example, the definition of a disk-shaped structuring element is easy in the continuous plane but especially on a small scale this is difficult or even impossible to realize conveniently on a discrete grid. To this end, it is necessary to specify continuous mathematical morphology from the angle of curve evolution. By this method, discrete mathematical morphology can be interpreted as the numerical implementation of a continuous-scale evolution.

According to [175], dilation can be performed at infinitesimal steps. This motion generates a set of velocity vectors, one for each point on the boundary of the disk-shaped (or more generally, convex) structuring element. For this purpose, it is possible to parameterize these vectors by the angle θ running over all possible angles about a central point in the plane, so that $\theta \in [0, 2\pi]$. For a given initial image $f := f(x, y)$, where (x, y) denotes a point in the image domain Ω , let $u := u(x, y, t)$ be the image evolving under the process of interest in time t . Then we have:

$$\partial_t u = \sup_{\theta} \{R(\theta) \cdot \nabla u\}, \quad (5.5)$$

where $R(\theta)$ is a function representing the boundary of the convex structuring element. In this way, the following velocities are obtained for popular structuring elements S :

$$\sup_{\theta} \{R(\theta) \cdot \nabla u\} = \begin{cases} \|\nabla u\|_1, & S = \text{diamond} \\ \|\nabla u\|_2, & S = \text{disk} \\ \|\nabla u\|_{\infty}, & S = \text{square} \end{cases} \quad (5.6)$$

Focusing again on the use of a disk-shaped structuring element and generalizing the process to include erosion, we obtain the PDEs for gray-value dilation (+) and erosion (−) as

$$\partial_t u = \pm \|\nabla u\|_2 = \pm \sqrt{(\partial_x u)^2 + (\partial_y u)^2} \quad \text{on} \quad \Omega \times (0, \infty) \quad (5.7)$$

which we supplement by *Neumann boundary conditions*:

$$\partial_n u = 0 \quad \text{on} \quad \partial\Omega \times (0, \infty) \quad (5.8)$$

and the initial condition defined by an input image f

$$u(x, y, 0) := f(x, y) \quad \forall (x, y) \in \Omega \quad (5.9)$$

While it is possible to describe already at this point a matrix-valued counterpart of the PDEs as in (5.7) as can be seen in [34, 35], we refrain from this here for shortness of presentation.

5.3 Numerical Methods for the PDEs of Dilation/Erosion

In this part, we briefly survey the schemes mentioned in the introduction that we will also consider here for realizing our PDE-based approach. These are the first-order accurate *Rouy-Tourin (RT)* scheme which is proposed in [170], the second-order method of *Osher and Sethian (OS)* [154], and as a state-of-the-art approach we consider the *flux corrected transport (FCT)* algorithm [32].

As a side note, we apply the symbol of $u_{i,j}^n$ as the gray-value of the evolving image u at the pixel located in the i^{th} row and j^{th} column of the image at the n^{th} time step during the morphological progress. We recall standard notations for backward and forward differences in x - and y -directions as follows:

$$\begin{aligned} D_-^x u_{i,j}^n &= u_{i,j}^n - u_{i-1,j}^n, & D_+^x u_{i,j}^n &= u_{i+1,j}^n - u_{i,j}^n, \\ D_-^y u_{i,j}^n &= u_{i,j}^n - u_{i,j-1}^n, & D_+^y u_{i,j}^n &= u_{i,j+1}^n - u_{i,j}^n \end{aligned} \quad (5.10)$$

Now we consider a uniform pixel width h in both spatial grid directions in an image and a numerical time step size τ for the evolution. Our aim is now to discretize the PDE (5.7), sticking thereby for the presentation here to the case of *dilation* with a disk-shaped structuring element. Then, in the *RT scheme*, the dilation operation is expressed by:

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{\tau}{h} \sqrt{(\max(0, D_+^x u_{i,j}^n, -D_-^x u_{i,j}^n))^2 + (\max(0, D_+^y u_{i,j}^n, -D_-^y u_{i,j}^n))^2} \quad (5.11)$$

while the second-order *OS method* is given by

$$u_{i,j}^{n+1} = \frac{u_{i,j}^n}{2} + \frac{u_{i,j}^{-n+1}}{2} + \frac{\tau}{2h} L(u^{-n+1}, i, j) \quad (5.12)$$

where

$$u_{i,j}^{-n+1} = u_{i,j}^n + \frac{\tau}{h} L(u^n, i, j) \quad (5.13)$$

and

$$\begin{aligned} L(u^n, i, j) = & \left[\left(\min \left\{ D_-^x u_{i,j}^n + \frac{1}{2} \text{mm}(D_-^x D_+^x u_{i,j}^n, D_-^x D_-^x u_{i,j}^n), 0 \right\} \right)^2 \right. \\ & + \left(\max \left\{ D_+^x u_{i,j}^n - \frac{1}{2} \text{mm}(D_+^x D_+^x u_{i,j}^n, D_-^x D_+^x u_{i,j}^n), 0 \right\} \right)^2 \\ & + \left(\min \left\{ D_-^y u_{i,j}^n + \frac{1}{2} \text{mm}(D_-^y D_+^y u_{i,j}^n, D_-^y D_-^y u_{i,j}^n), 0 \right\} \right)^2 \\ & \left. + \left(\max \left\{ D_+^y u_{i,j}^n - \frac{1}{2} \text{mm}(D_+^y D_+^y u_{i,j}^n, D_-^y D_+^y u_{i,j}^n), 0 \right\} \right)^2 \right]^{\frac{1}{2}} \end{aligned} \quad (5.14)$$

The function $\text{mm}(\cdot, \cdot)$ indicates the *minmod function* which is given as

$$\text{mm}(\alpha, \beta) := \begin{cases} \max(\alpha, \beta), & \alpha < 0, \beta < 0 \\ \min(\alpha, \beta), & \alpha > 0, \beta > 0 \\ 0, & \text{otherwise} \end{cases} \quad (5.15)$$

In this part we provide a brief introduction of the *FCT scheme*. The main concept in the FCT scheme is to use the RT scheme in a *predictor step* in a first phase. Then the unwanted blurring effects generated by the first-order upwind derivatives in the RT scheme are measured to reverse the associated quantity in a *corrector step* that performs stabilized

inverse diffusion. As the next step, we write the values obtained after the predictor step performed by the RT scheme in the format $u_{i,j}^p$ at pixel (i, j) . With the definitions :

$$g_{i+1/2,j} := \text{mm} \left(D_-^x u_{i,j}^p, \frac{\tau}{2h} D_+^x u_{i,j}^p, D_+^x u_{i+1,j}^p \right), \quad (5.16)$$

$$g_{i,j+1/2} := \text{mm} \left(D_-^y u_{i,j}^p, \frac{\tau}{2h} D_+^y u_{i,j}^p, D_+^y u_{i,j+1}^p \right), \quad (5.17)$$

where $\text{mm}(\cdot, \cdot, \cdot)$ is a straightforward extension of (5.15) and

$$Q_h := \sqrt{\left(\frac{\tau}{2h} |u_{i+1,j}^p - u_{i-1,j}^p| \right)^2 + \left(\frac{\tau}{2h} |u_{i,j+1}^p - u_{i,j-1}^p| \right)^2}, \quad (5.18)$$

$$Q_l := \sqrt{(\delta u_i^p)^2 + (\delta u_j^p)^2}, \quad (5.19)$$

where the stabilized inverse diffusive fluxes are given by

$$u_i^p := \frac{\tau}{2h} |u_{i+1,j}^p - u_{i-1,j}^p| + g_{i+1/2,j} - g_{i-1/2,j}, \quad (5.20)$$

$$u_j^p := \frac{\tau}{2h} |u_{i,j+1}^p - u_{i,j-1}^p| + g_{i,j+1/2} - g_{i,j-1/2}, \quad (5.21)$$

we can write the subsequent *corrector step* of the FCT scheme as

$$u_{i,j}^{n+1} = u_{i,j}^p + Q_h - Q_l \quad (5.22)$$

To summarize, a subsequent application of scheme (5.11) for obtaining predicted data $u_{i,j}^p$ – instead of $u_{i,j}^{n+1}$ in (5.11) – and the corrector step (5.22) making use of the predicted values is equivalent to the FCT scheme.

Finally, our aim is to work with fields of symmetric 2×2 matrices which represent color data instead of gray-values. For the definition of corresponding numerical schemes, we proceed in a straightforward fashion building upon (5.10)–(5.22). Instead of the evolving gray-values $u_{i,j}^n$ we will plug in the 2×2 matrices $U_{i,j}^n$, with $U_{i,j}^0 := f_{i,j}$ where f corresponds to a given color image. This implicitly defines underlying color-valued PDEs. Obviously, in order to give a meaning to the formulae (5.10)–(5.22) in the latter setting, we must define suitable notions for maximum and minimum of up to three matrices, and we must

give useful expressions for the square root and the absolute value of occurring matrices. This will be done in Section 5.5.

5.4 Color Images and Matrix Fields

In this section, we shortly recall the conversion of rgb values to matrices as in [36]. Given an rgb image we transform it in two steps into a matrix field of equal dimensions, i.e. we assign each pixel of the image a symmetric 2×2 matrix. In the first step, we transform the rgb color values to the hcl color space, assuming that red, green and blue intensities are normalized to $[0, 1]$. For a pixel with such intensities r, g, b , we obtain its hue h , chroma c and luminance l via $M = \max\{r, g, b\}$, $m = \min\{r, g, b\}$, $c = M - m$, $l = \frac{1}{2}(M + m)$, and $h = \frac{1}{6}(g - b)/M$ modulo 1 if $M = r$, $h = \frac{1}{6}(b - r)/M + \frac{1}{3}$ if $M = g$, $h = \frac{1}{6}(r - g)/M + \frac{2}{3}$ if $M = b$, cf. [7].

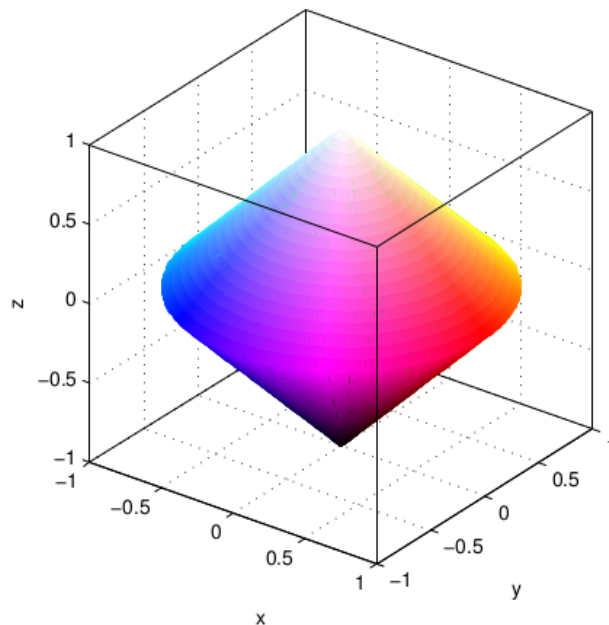


Figure 5-1: Color bi-cone, figure adapted from [36]

Replacing then luminance l with $\tilde{l} := 2l - 1$, and interpreting c , $2\pi h$, and \tilde{l} as radial, angular and axial coordinates of a cylindrical coordinate system, we have a bijection from the unit cube of triples (r, g, b) onto a solid bi-cone, see Figure 5-1.

The bi-cone is then transformed to the Cartesian coordinates via $x = c \cos(2\pi h)$, $y = c \sin(2\pi h)$, $z = \tilde{l}$. The second step takes the coordinates (x, y, z) and maps them to symmetric matrices $A \in \text{Sym}(2)$ via

$$A := \frac{\sqrt{2}}{2} \begin{pmatrix} z - y & x \\ x & z + y \end{pmatrix} \quad (5.23)$$

It is worthwhile noting that the mapping $\Psi : \mathbb{R}^3 \rightarrow \text{Sym}(2)$ in (5.23) is bijective. Denoting by $\mathcal{M} \subset \text{Sym}(2)$ the set of all matrices A that correspond to points of the bi-cone, we have in fact by (5.23) a bijection between the *rgb* color space and the bi-cone \mathcal{M} . The inverse transform is obtained in a straightforward way, cf. [36].

5.5 Pseudo Supremum/Infimum and Functions of Matrices

As indicated at the end of Section 5.3, we need to give meaning to the maximum and minimum of up to three matrices of $\text{Sym}(2)$, as well as to the square root and the absolute value of such matrices. Thereby we rely on corresponding notions as discussed e.g. in [34].

A related point to consider is that any matrix $A \in \text{Sym}(2)$ can be decomposed into the format $A = V \text{diag}(\lambda_1, \lambda_2) V^\top$ where $V := (v_1, v_2)$ accumulates the eigenvectors v_1, v_2 of A as column vectors and $\lambda_{1,2}$ denote the corresponding eigenvalues. Then one may define a function φ of a matrix A via

$$\varphi(A) := V \text{diag}(\varphi(\lambda_1), \varphi(\lambda_2)) V^\top \quad (5.24)$$

in terms of its standard scalar representation. With $\varphi(\cdot) = \sqrt{\cdot}$ and $\varphi(\cdot) = |\cdot|$ we thus obtain square root and absolute value of a symmetric matrix, respectively.

Regarding the formula of numerical schemes in Section 5.3, we need to calculate the maximum and minimum of up to three symmetric matrices. It will turn out that instead of

maximum and minimum we will seek a supremum and infimum, respectively, and it will suffice to elaborate in detail on the supremum.

Concerning matrices $A, B, C \in \text{Sym}(2)$, determining the supremum of two such matrices can be done making use of (5.24) by

$$\sup(A, B) := \frac{A + B}{2} + \frac{|A - B|}{2} \quad (5.25)$$

adopting a corresponding scalar relation. Obviously, we can proceed by

$$\begin{aligned} \sup_1 &:= \sup(A, \sup(B, C)), \\ \sup_2 &:= \sup(B, \sup(A, C)), \\ \sup_3 &:= \sup(C, \sup(A, B)) \end{aligned} \quad (5.26)$$

But generally, for $A, B, C \in \text{Sym}(2)$ we have

$$\sup_1 \neq \sup_2 \neq \sup_3 \neq \sup_1 \quad (5.27)$$

Consequently, we approximate the supremum of $\{A, B, C\}$ by calculating the average of \sup_1, \sup_2 and \sup_3 , as the \sup_{avg} which is an upper bound of each initial matrix.

To improve this often very generous upper bound, we find the optimal value of $\eta \geq 0$ in such a way that

$$\sup_{avg} - \eta I \geq W, \quad W \in \{A, B, C\}, \quad (5.28)$$

where I is the 2×2 identity matrix. The optimal amount η_{opt} of η in (5.28) is the minimum eigenvalue of $(\sup_{avg} - A)$, $(\sup_{avg} - B)$, and $(\sup_{avg} - C)$. At the end of this process, we obtain a proper supremum of $\{A, B, C\}$ as

$$\sup_{opt}(A, B, C) := \sup_{avg} - \eta_{opt} I \quad (5.29)$$

To obtain an infimum of three matrices, one may simply set

$$\inf_{opt}(A, B, C) := \sup_{opt}(-A, -B, -C) \quad (5.30)$$

5.6 Experimental Results

As the first experiment we test if our color morphology operations retrieve gray-scale morphology, since this may be considered a necessary condition to obtain a reasonable extension of the latter. To this end, we employ *rgb* values for black and white and use the new color-valued FCT scheme as described in Section 5.3. In Fig. 5-2 we exhibit the result of dilation/erosion on the *yin-yang* image of size 256×256 . Operations are performed ten times with time step size $\tau = 1/2$ and the disk-shaped structuring element. As illustrated, outcomes are equivalent to gray-scale morphology.

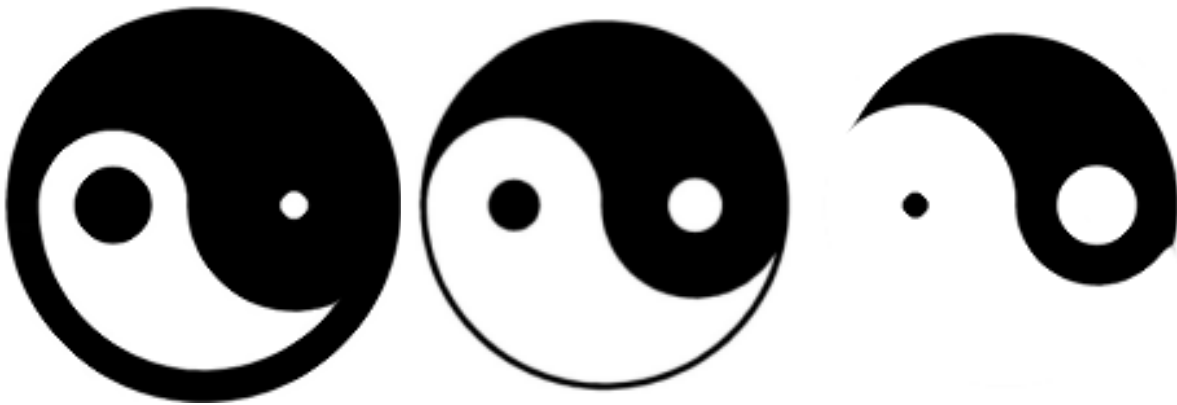


Figure 5-2: (Center) Input image yin-yang [5] defined using *rgb* values for black/white. (Left) Ten times dilation with color-valued FCT. (Right) Ten times erosion with color-valued FCT.

In our second test we aim to observe dilation and erosion in color space with the RT scheme. The reasoning is here, that independently from its usefulness by its own the RT scheme serves as the basis of the FCT method and it is very similar to the first-order method that the OS scheme builds upon. Therefore, it is of fundamental importance for our PDE-based approach that the RT scheme yields reasonable results, as otherwise the more advanced OS and FCT schemes cannot be expected to do something valuable.

As observed in Fig. 5-3, we can confirm that the RT scheme performs as expected. Taking the classic *Lena* test image of resolution 128×128 as input image, we see that after six iterations of dilation and erosion with time step size $\tau = 1/2$ that bright and dark colors are enhanced, respectively. The blurring we observe here is the standard numerical artifacts resulting from the first-order upwind discretization. In an extension of these experiments,



Figure 5-3: (Top row) Original *Lenna* image [2], and results of dilation and erosion computed with the RT scheme. (Bottom row) Morphological Laplacian and results of five and ten iterations of shock filtering with the RT scheme.

we compute the morphological Laplacian and show results of shock filtering based on our framework using also the RT scheme with $\tau = 1/2$. As observed, we obtain visually very plausible results for this process. It is important to note that for the purpose of shock filtering the RT scheme is the optimal PDE-based method since the shock-filtering process is designed to give sharp edges.

Our next experiment serves two purposes. On the one hand we compare the quality of the numerical schemes RT, OS, and FCT in our new framework in order to see if the non-linear operations performed in the algorithms still give reasonable, interpretable results. On the other hand, we compare here with the method of Burgeth and Kleefeld (BK) [36] that is technically more similar to classic, lattice-based morphology than our PDE-based schemes. The BK method employs the same color space yet with a different means of addition and subtraction of color matrices. As a side note, we employ in BK a cross-shaped structuring element here as the approximation of a disk on a 3×3 grid.

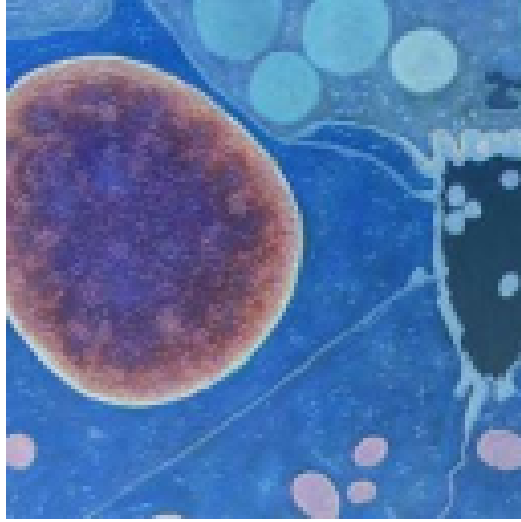


Figure 5-4: Input image for comparison of numerical methods [29]

To this end, we employ a test image based on a micro biological scene, based on an oil painting of Carolyn K. Snyder. It is of resolution 128×128 and features diverse colors as well as round structures, see Fig. 5-4.

The results of our comparison are displayed in Fig. 5-5 where we show the images after several dilation steps. They show that all of the PDE-based methods give results of expected quality. The RT scheme yields a blurry dilated image and the FCT scheme very sharp edges while the OS method is somewhere in between those schemes. We also see no obvious color distortions, and round shapes evolve in a round way as by the underlying disk-shaped structuring element used for the PDE-based methods. In the result of the BK method for discrete morphology, we recognize the influence of the cross-shaped structuring element while we do not observe other color effects as in the results of the PDE-based schemes, although these employ different addition and subtraction rules. Note that the PDE-based FCT method gives visually as sharp edges as the BK method.

Our next and final experiment is dealing with the influence of the color space. Obviously, the value of the pseudo supremum resp. infimum of two colors in the dilation resp. erosion process is dependent on the location of those colors in the bi-conal color structure. Generally, the pseudo supremum of any color faced with white is white and the pseudo infimum of any color faced with black is black. Thus we will not see any new appearing color at the edge to a black or white region.

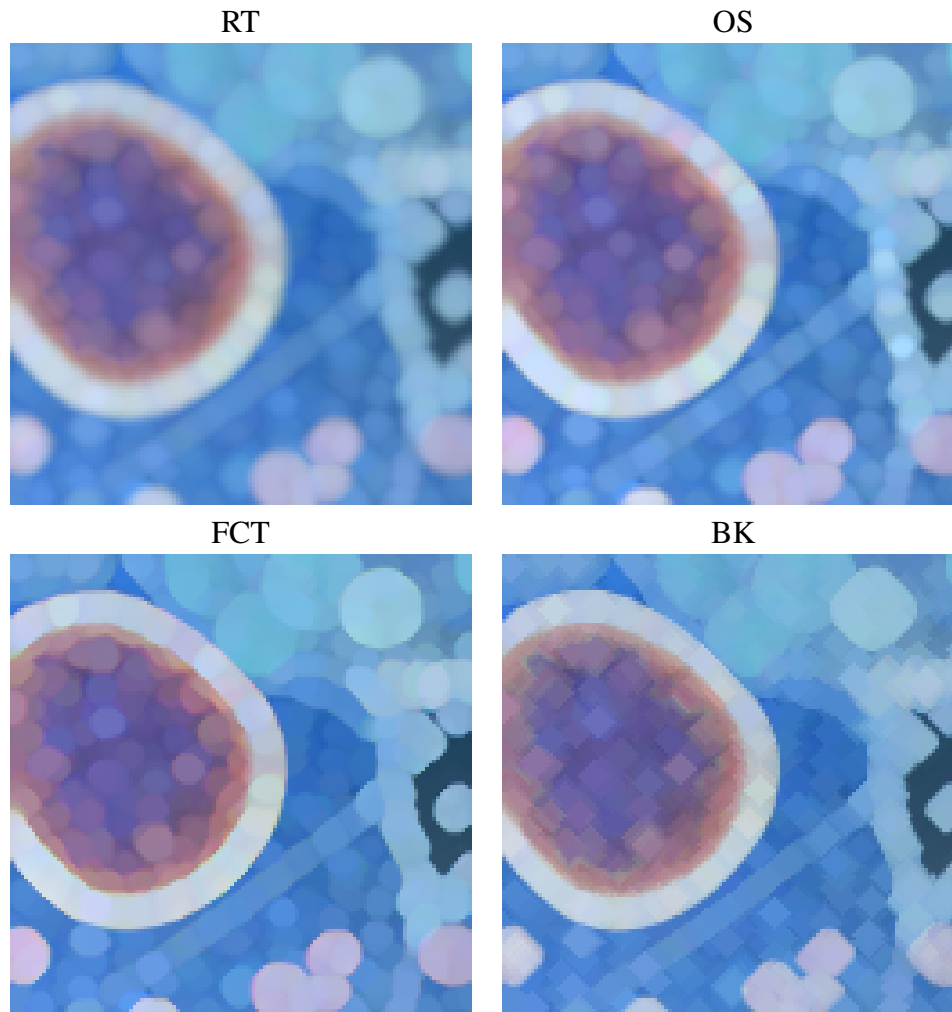


Figure 5-5: Results of eight time steps dilation with $\tau = 1/2$ for indicated PDE-based schemes and in accordance four times erosion of BK method [29].

Also in other cases, if one of the primary colors equals the pseudo supremum or infimum of them, then we do not have any color changes in the border of those colors during basic morphological operations. Some examples of this situation are indicated in Fig. 5-6 by yellow frames. But, if the pseudo supremum or infimum of the two colors equals another color, it appears as a modified color. Some situations like these are marked with black frames in Fig. 5-6. Note that the new colors are not false colors [181] but appear as more resp. less saturated versions of bordering colors.

It is worthwhile to investigate this phenomenon at hand of an example dealing with the colors *light magenta* (*lm*), *dark magenta* (*dm*) and *cyan* (*cy*). Light magenta has the *rgb*

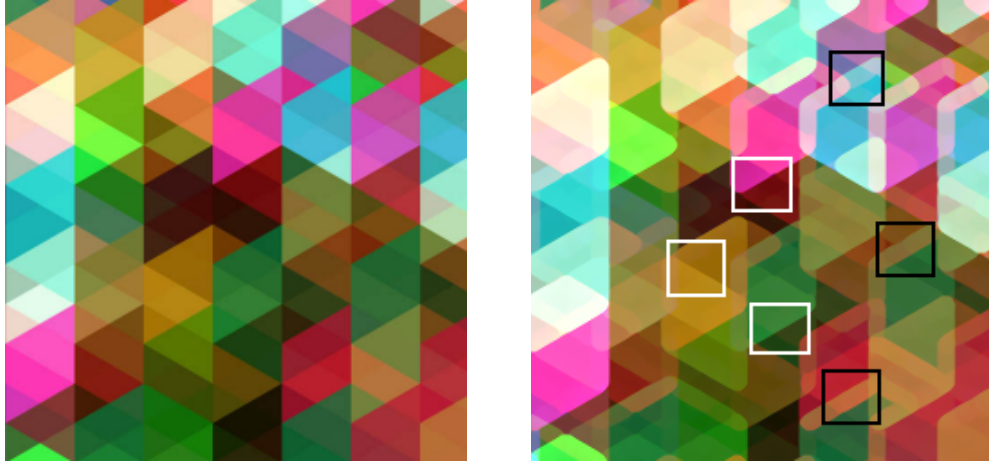


Figure 5-6: Original image of size 256×256 (left) and result after five iterations of dilation using FCT with $\tau = 1/2$ and a disk-shaped structuring element (right). Black frames indicate new colors that appear by use of the supremum rule and the yellow ones mark color interaction without new colors [29].

values $(252, 58, 157)$, the numbers are $(217, 57, 153)$ for dark magenta, and $(62, 186, 212)$ for cyan. The equivalent matrices of these colors are as follows (entries rounded):

$$\mathbf{lm} = \begin{bmatrix} 0.427 & 0.463 \\ 0.463 & -0.122 \end{bmatrix}, \quad \mathbf{dm} = \begin{bmatrix} 0.314 & 0.359 \\ 0.359 & -0.208 \end{bmatrix}, \quad \mathbf{cy} = \begin{bmatrix} 0.128 & -0.409 \\ -0.409 & -0.022 \end{bmatrix} \quad (5.31)$$

By computing corresponding pseudo suprema of two matrices, we obtain (rounded):

$$\sup(\mathbf{lm}, \mathbf{dm}) = \begin{bmatrix} 0.429 & 0.461 \\ 0.461 & -0.119 \end{bmatrix}, \quad \sup(\mathbf{dm}, \mathbf{cy}) = \begin{bmatrix} 0.616 & -0.025 \\ -0.025 & 0.280 \end{bmatrix} \quad (5.32)$$

The *rgb* amounts of the first pseudo supremum are $(252, 59, 157)$, while in the second one we gain $(200, 178, 239)$.



Figure 5-7: Colors in the example [29]

These observations show that during a dilation process, for the left inner edge as seen in Fig. 5-7, we have a color almost like light magenta which appears at the border as the extension of the light magenta color, while in the right inner border, a new color emerges as the supremum of the dark magenta and the cyan areas. However, observe also here that this is not a false color.

Chapter 6

General Conclusion

6.1 Summary of Contributions

In this thesis, we investigated two different areas of the deep learning research.

In chapter 3, we introduced a novel hybrid deep learning architecture for interactive region segmentation and captioning whereby the user is able to specify an arbitrary region of the image that should be highlighted and described. To this end, we trained three different types of the Fully Convolutional Network (FCN) as various versions of our interactive segmentation module to identify the User Intended Region (UIR). In parallel, a Recognition and Captioning (ReCap) module is used to understand the visual contents of the scene by drawing bounding boxes around detected objects, estimating their objectness scores and producing their linguistic descriptions. During our fusion approach, the detected UIR will be explained with the caption of the best match bounding box. To the best of our knowledge, this is the first work that provides such a comprehensive output. In addition, replacement of the bounding boxes with the result of the interactive segmentation leads to a better understanding of the image captioning output as well as an enhancement in object localization accuracy.

The proposed model can be used for mapping natural language to images and vice versa, better control of visual contents in social media, medical image understanding and providing a better understanding of the virtual world for visually impaired people.

In chapter 4, we proposed a bidirectional relation between deep texture representation and

its linguistic description via a hybrid CNN-RNN model that enables end-to-end learning of the selective texture labeling. This novel architecture provides new opportunities to describe, search, and also retrieve texture images by their linguistic descriptions. To be able to train such a model, we generated a multi-label texture data set that covers color, material, and pattern attributes of the textured images simultaneously. Our contribution to the automatic generation of texture descriptions provides an opportunity to enrich the existing vocabulary of the image captioning task. Such a conceptual extension can be used in the fine-grained image captioning as well as the design of more intelligent recommender systems.

In the final chapter, we proposed a novel approach to define mathematical morphology on color images. To this end, we converted common RGB-values of the color images into a new biconal color space and then combined two approaches of mathematical morphology to give meaning to the maximum and the minimum of the matrix field data and formulate our novel strategy.

6.2 Future Directions

Combination of the interactive segmentation module with the pretrained CNN of the recognition and captioning module: In chapter 3, two modules of the proposed architecture should be trained separately on different types of the training data. As the structure of the interactive segmentation module and the pretrained CNN of the recognition and captioning module are identical, it would be interesting to develop a new strategy for their combination. This could be useful to have a seamless architecture that can be trained once. To this aim, the training data should involve user interactions that are able to influence segmentation, recognition and captioning tasks simultaneously.

Design of a recommender system based on the provided deep texture representation: Our new understanding about texture representations and their linguistic explanations via deep learning makes it possible to design a new generation of AI-based recommender systems that are able to interact with users of the commercial websites to introduce produc-

tions that satisfy customer's criteria about the color, material and the intrinsic pattern of the products such as clothes, building stones, leather products, variety of fabrics, artworks and wooden products.

Bibliography

- [1] Caffe2, a new lightweight, modular, and scalable deep learning framework. <https://caffe2.ai/>. Accessed: 23.11.2017.
- [2] Image of lena söderberg used in many image processing experiments. [https://en.wikipedia.org/wiki/Lenna#/media/File:Lenna_\(test_image\).png](https://en.wikipedia.org/wiki/Lenna#/media/File:Lenna_(test_image).png). Accessed: 21.1.2018.
- [3] The picture is publicly available via license: CC BY-NC 4.0. <https://creativecommons.org/licenses/by-nc/4.0/>. Accessed: 21.1.2018.
- [4] Structure of a biological neuron. https://www.wpclipart.com/medical/anatomy/nervous_system/neuron/neuron.png.html. Accessed: 21.1.2018.
- [5] Yin and yang. https://en.wikipedia.org/wiki/Yin_and_yang#/media/File:Yin_yang.svg. Accessed: 21.1.2018.
- [6] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. <https://www.tensorflow.org/>, 2015. Software available from tensorflow.org.
- [7] M. K. Agoston. *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. Springer, 2005.
- [8] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–2041, 2006.
- [9] N. Ahuja. Dot pattern processing using voronoi neighborhoods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (3):336–343, 1982.
- [10] H. Akaike. Fitting autoregressive models for prediction. *Annals of the institute of Statistical Mathematics*, 21(1):243–247, 1969.
- [11] R. Al-Rfou et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.
- [12] B. Alexe, T. Deselaers, and V. Ferrari. What is an object? In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 73–80. IEEE, 2010.

- [13] S. Alpert, M. Galun, A. Brandt, and R. Basri. Image segmentation by probabilistic bottom-up aggregation and cue integration. In *Proc. CVPR*, 2007.
- [14] L. Alvarez, F. Guichard, P. L. Lions, and J. M. Morel. Axioms and fundamental equations of image processing. *Archive for rational mechanics and analysis*, 123(3):199–257, 1993.
- [15] V. Andrearczyk and P. F. Whelan. Using filter banks in convolutional neural networks for texture classification. *Pattern Recognition Letters*, 84:63–69, 2016.
- [16] E. Aptoula and S. Lefèvre. A comparative study on multivariate mathematical morphology. *Pattern Recognition*, 40(11):2914–2929, 2007.
- [17] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 328–335, 2014.
- [18] A. B. Arehart, L. Vincent, and B. B. Kimia. Mathematical morphology: The hamilton-jacobi connection. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, pages 215–219. IEEE, 1993.
- [19] H. Autrum et al. *Central Processing of Visual Information A: Integrative Functions and Comparative Data*. Springer Science & Business Media, 2012.
- [20] X. Bai and G. Sapiro. Geodesic matting: A framework for fast interactive image and video segmentation and matting. *International journal of computer vision*, 82(2):113–132, 2009.
- [21] 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. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 992–999. IEEE, 2016.
- [22] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer vision—ECCV 2006*, pages 404–417, 2006.
- [23] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
- [24] D. Bell. *The coming of post-industrial society: A venture in social forecasting*. Basic Books, New York, 1973.
- [25] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database (supplemental material). In *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.
- [26] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

- [27] L. J. Borg-Graham, C. Monier, and Y. Fregnac. Visual input evokes transient and strong shunting inhibition in visual cortical neurons. *Nature*, 393(6683):369, 1998.
- [28] A. S. Boroujerdi. *Self depicted figures and self taken photographs*. 2018.
- [29] A. S. Boroujerdi, M. Breuß, B. Burgeth, and A. Kleefeld. PDE-based color morphology using matrix fields. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 461–473. Springer, 2015.
- [30] A. S. Boroujerdi, M. Khanian, and M. Breuß. Deep interactive region segmentation and captioning. In *Signal Image Technology and Internet-based Systems (SITIS), 13th International Conference on*. IEEE, 2017.
- [31] Y. Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Proc. ICCV*, 2001.
- [32] M. Breuß and J. Weickert. A shock-capturing algorithm for the differential equations of dilation and erosion. *Journal of Mathematical Imaging and Vision*, 25(2):187–201, 2006.
- [33] R. W. Brockett and P. Maragos. Evolution equations for continuous-scale morphology. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 3, pages 125–128. IEEE, 1992.
- [34] B. Burgeth, M. Breuß, S. Didas, and J. Weickert. PDE-based morphology for matrix fields: Numerical solution schemes. *Tensors in Image Processing and Computer Vision*, pages 125–150, 2009.
- [35] B. Burgeth, A. Bruhn, S. Didas, J. Weickert, and M. Welk. Morphology for matrix data: Ordering versus PDE-based approach. *Image and Vision Computing*, 25(4):496–511, 2007.
- [36] B. Burgeth and A. Kleefeld. An approach to color-morphology based on einstein addition and loewner order. *Pattern Recognition Letters*, 47:29–39, 2014.
- [37] G. J. Burghouts and J. M. Geusebroek. Material-specific adaptation of color invariant features. *Pattern Recognition Letters*, 30(3):306–313, 2009.
- [38] T. Çelik et al. CSS color module level 3. 2011.
- [39] R. Chellappa and S. Chatterjee. Classification of textures using gaussian markov random fields. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(4):959–963, 1985.
- [40] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.
- [41] X. Chen and C. L. Zitnick. Mind’s eye: A recurrent visual representation for image caption generation. In *Proc. CVPR*, 2015.

- [42] F. Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [43] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 3606–3613. IEEE, 2014.
- [44] M. Cimpoi, S. Maji, I. Kokkinos, and A. Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *International Journal of Computer Vision*, 118(1):65–94, 2016.
- [45] M. Clark, A. C. Bovik, and W. S. Geisler. Texture segmentation using gabor modulation/demodulation. *Pattern Recognition Letters*, 6(4):261–267, 1987.
- [46] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
- [47] M. L. Comer and E. J. Delp. Morphological operations for color image processing. *J. Electronic Imaging*, 8(3):279–289, 1999.
- [48] A. Criminisi, T. Sharp, and A. Blake. Geos: Geodesic image segmentation. *Computer Vision—ECCV 2008*, pages 99–112, 2008.
- [49] N. Cristianini, J. Shawe-Taylor, and H. Lodhi. Latent semantic kernels. *Journal of Intelligent Information Systems*, 18(2):127–152, 2002.
- [50] G. R. Cross and A. K. Jain. Markov random field texture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):25–39, 1983.
- [51] 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*, volume 1, pages 1–2. Prague, 2004.
- [52] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.
- [53] 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)*, 18(1):1–34, 1999.
- [54] M. Denkowski and A. Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380, 2014.
- [55] J. Devlin, H. Cheng, H. Fang, S. Gupta, L. Deng, X. He, G. Zweig, and M. Mitchell. Language models for image captioning: The quirks and what works. *arXiv preprint arXiv:1505.01809*, 2015.
- [56] D. V. Dijk, T. Teräsvirta, and P. H. Franses. Smooth transition autoregressive models—a survey of recent developments. *Econometric reviews*, 21(1):1–47, 2002.

- [57] 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*, pages 2625–2634, 2015.
- [58] M. Drozdal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal. The importance of skip connections in biomedical image segmentation. In *International Workshop on Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*, pages 179–187. Springer, 2016.
- [59] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proc. ICCV*, 2015.
- [60] D. Elliott and F. Keller. Image description using visual dependency representations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1292–1302, 2013.
- [61] D. Elliott, V. Lavrenko, and F. Keller. Query-by-example image retrieval using visual dependency representations. In *International Conference on Computational Linguistics*, 2014.
- [62] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.
- [63] B. Fan, Z. Wang, and F. Wu. *Local Image Descriptor: Modern Approaches*. Springer, 2015.
- [64] H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig. From captions to visual concepts and back. In *Proc. CVPR*, 2015.
- [65] A. Farhadi et al. Every picture tells a story: Generating sentences from images. In *European conference on computer vision*, pages 15–29. Springer, 2010.
- [66] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.
- [67] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.
- [68] M. A. Georgeson. Spatial fourier analysis and human vision. *Tutorial essays in psychology*, 2:39–88, 1979.
- [69] R. Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

- [70] 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*, pages 580–587, 2014.
- [71] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
- [72] J. Goutsias, H. J. Heijmans, and K. Sivakumar. Morphological operators for image sequences. *Computer Vision and Image Understanding*, 62(3):326–346, 1995.
- [73] L. Grady. Random walks for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 28(11):1768–1783, 2006.
- [74] V. Gulshan, C. Rother, A. Criminisi, A. Blake, and A. Zisserman. Geodesic star convexity for interactive image segmentation. In *Proc. CVPR*, 2010.
- [75] A. Gupta et al. Choosing linguistics over vision to describe images. In *AAAI*, page 1, 2012.
- [76] C. Han, E. Risser, R. Ramamoorthi, and E. Grinspun. Multiscale texture synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 51. ACM, 2008.
- [77] R. M. Haralick et al. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, (6):610–621, 1973.
- [78] E. Hayman, B. Caputo, M. Fritz, and J. O. Eklundh. On the significance of real-world conditions for material classification. In *European conference on computer vision*, pages 253–266. Springer, 2004.
- [79] D. He and L. Wang. Texture unit, texture spectrum, and texture analysis. *IEEE transactions on Geoscience and Remote Sensing*, 28(4):509–512, 1990.
- [80] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, pages 346–361. Springer, 2014.
- [81] 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*, pages 770–778, 2016.
- [82] M. Hilbert and P. López. The world’s technological capacity to store, communicate, and compute information. *Science*, 2011.
- [83] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [84] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

- [85] M. Hodosh, P. Young, and J. Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.
- [86] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [87] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.
- [88] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.
- [89] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat’s striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.
- [90] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.
- [91] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Proc. NIPS*, 2015.
- [92] 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*, pages 3304–3311. IEEE, 2010.
- [93] S. Jégou, M. Drozdal, D. Vazquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1175–1183. IEEE, 2017.
- [94] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- [95] J. Johnson, A. Karpathy, and L. Fei-Fei. Denscap: Fully convolutional localization networks for dense captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4565–4574, 2016.
- [96] J. L. Jones, X. Xie, and E. Essa. Image segmentation using combined user interactions. In *Proc. CVMC*, 2013.
- [97] J. M. Joyce. Kullback-leibler divergence. In *International Encyclopedia of Statistical Science*, pages 720–722. Springer, 2011.
- [98] R. E. Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 1(2):152–192, 1963.

- [99] A. Karbasi, S. S. Laskukalayeh, and S. M. Fahimifard. Comparison of nnarx, ann and arima techniques to poultry retail price forecasting. *International Association of Agricultural Economists*, pages 16–22, 2009.
- [100] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.
- [101] A. Karpathy, A. Joulin, and L. Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems*, pages 1889–1897, 2014.
- [102] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [103] R. Kiros, R. Salakhutdinov, and R. S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*, 2014.
- [104] P. Komarek. Logistic regression for data mining and high-dimensional classification. Pittsburgh, PA, 2004.
- [105] J. Krause, J. Johnson, R. Krishna, and L. Fei-Fei. A hierarchical approach for generating descriptive image paragraphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3337–3345. IEEE, 2017.
- [106] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. J. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, pages 1–42, 2017.
- [107] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [108] A. Kuleshov and A. Bernstein. *Incremental Construction of Low-Dimensional Data Representations*, pages 55–67. Springer International Publishing, 2016.
- [109] G. Kulkarni et al. Babytalk: Understanding and generating simple image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2891–2903, 2013.
- [110] P. Kuznetsova, V. Ordonez, A. C. Berg, T. L. Berg, and Y. Choi. Collective generation of natural image descriptions. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 359–368. Association for Computational Linguistics, 2012.
- [111] G. Kylberg. *Kylberg Texture Dataset v. 1.0*. Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, 2011.

- [112] K. I. Laws. Rapid texture identification. In *Image processing for missile guidance*, volume 238, pages 376–382. International Society for Optics and Photonics, 1980.
- [113] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278, 2005.
- [114] Y. LeCun. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [115] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [116] S. Lefebvre and H. Hoppe. Appearance-space texture synthesis. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 541–548. ACM, 2006.
- [117] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textures. *International journal of computer vision*, 43(1):29–44, 2001.
- [118] S. Li, G. Kulkarni, T. L. Berg, A. C. Berg, and Y. Choi. Composing simple image descriptions using web-scale n-grams. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 220–228. Association for Computational Linguistics, 2011.
- [119] Y. Li, J. Sun, C. K. Tang, and H. Y. Shum. Lazy snapping. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 303–308. ACM, 2004.
- [120] D. Lin, J. Dai, J. Jia, K. He, and J. Sun. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3159–3167, 2016.
- [121] H. C. Lin, L. L. Wang, and S. N. Yang. Extracting periodicity of a regular texture based on autocorrelation functions. *Pattern recognition letters*, 18(5):433–443, 1997.
- [122] T. Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common objects in context. *arXiv preprint*, (arXiv: 1405.0312), 2014.
- [123] T. Y. Lin and S. Maji. Improved bilinear pooling with CNNs. *arXiv preprint arXiv:1707.06772*, 2017.
- [124] 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*, pages 1449–1457, 2015.
- [125] T. Lindeberg and J. Garding. Shape from texture from a multi-scale perspective. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, pages 683–691. IEEE, 1993.

- [126] X. Liu and D. Wang. Image and texture segmentation using local spectral histograms. *IEEE Transactions on Image Processing*, 15(10):3066–3077, 2006.
- [127] Z. Liu, X. Li, P. Luo, C. C. Loy, and X. Tang. Semantic image segmentation via deep parsing network. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1385, 2015.
- [128] S. Livens, P. Scheunders, G. Van de Wouwer, and D. Van Dyck. Wavelets for texture analysis, an overview. 1997.
- [129] A. Lobay and D. A. Forsyth. Shape from texture without boundaries. *International Journal of Computer Vision*, 67(1):71–91, 2006.
- [130] M. London and M. Häusser. Dendritic computation. *Annu. Rev. Neurosci.*, 28:503–532, 2005.
- [131] 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*, pages 3431–3440, 2015.
- [132] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. IEEE, 1999.
- [133] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- [134] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). In *Proc. ICLR*, 2015.
- [135] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille. Explain images with multimodal recurrent neural networks. *arXiv preprint*, (arXiv: 1410.1090), 2014.
- [136] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, 2001.
- [137] R. Mason and E. Charniak. Nonparametric method for data-driven image captioning. In *ACL (2)*, pages 592–598, 2014.
- [138] G. Matheron. *Éléments pour une théorie des milieux poreux*. 1967.
- [139] P. H. Matthews. *Syntactic Relations: A Critical Survey*. Cambridge Studies in Linguistics. Cambridge University Press, 2007.
- [140] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [141] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

- [142] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [143] M. Minsky, S. A. Papert, and L. Bottou. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.
- [144] M. Mitchell et al. Midge: Generating image descriptions from computer vision detections. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 747–756. Association for Computational Linguistics, 2012.
- [145] C. Napoles, M. Gormley, and B. Van Durme. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, pages 95–100. Association for Computational Linguistics, 2012.
- [146] A. Nenkova and L. Vanderwende. The impact of frequency on summarization. *Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-2005*, 101, 2005.
- [147] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.
- [148] A. Noma, A. B. V. Graciano, R. M. Cesar Jr, L. A. Consularo, and I. Bloch. Interactive image segmentation by matching attributed relational graphs. *Pattern Recognition*, 45(3):1159 – 1179, 2012.
- [149] T. Ojala, T. Maenpaa, M. Pietikainen, J. Viertola, J. Kyllonen, and S. Huovinen. Outex-new framework for empirical evaluation of texture analysis algorithms. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 1, pages 701–706. IEEE, 2002.
- [150] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.
- [151] V. Ojansivu and J. Heikkilä. Blur insensitive texture classification using local phase quantization. In *International conference on image and signal processing*, pages 236–243. Springer, 2008.
- [152] L. G. M. Ortiz, C. Wolff, and M. Lapata. Learning to interpret and describe abstract scenes. In *HLT-NAACL*, pages 1505–1515, 2015.
- [153] S. Osher and L. I. Rudin. Feature-oriented image enhancement using shock filters. *SIAM Journal on Numerical Analysis*, 27(4):919–940, 1990.
- [154] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

- [155] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.
- [156] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.
- [157] A. Paszke, S. Gross, S. Chintala, and G. Chanan. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. <http://pytorch.org/>, 2017. Accessed: 23.11.2017.
- [158] 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*, pages 1–8. IEEE, 2007.
- [159] R. W. Picard, I. M. Elfadel, and A. P. Pentland. Markov/gibbs texture modeling: aura matrices and temperature effects. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 371–377. IEEE, 1991.
- [160] Cisco Public. Cisco visual networking index: Forecast and methodology. <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf>. Accessed 30.10.2017.
- [161] L. Rabiner and B. Juang. An introduction to hidden markov models. *iee assp magazine*, 3(1):4–16, 1986.
- [162] T. Randen and J. H. Husoy. Multichannel filtering for image texture segmentation. *Optical Engineering*, 33(8):2617–2626, 1994.
- [163] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [164] D. Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, pages 827–832, 2015.
- [165] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *Proc. CVPR*, 2009.
- [166] B. Roark, M. Saraclar, and M. Collins. Discriminative n-gram language modeling. *Computer Speech & Language*, 21(2):373–392, 2007.
- [167] N. Rochester, J. Holland, L. Haibt, and W. Duda. Tests on a cell assembly theory of the action of the brain using a large digital computer. *IRE Transactions on Information Theory*, 2(3):80–93, 1956.

- [168] 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*, 32(1):105–119, 2010.
- [169] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004.
- [170] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, 1992.
- [171] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.
- [172] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [173] C. Sagiv, N. A. Sochen, and Y. Y. Zeevi. Integrated active contours for texture segmentation. *IEEE transactions on image processing*, 15(6):1633–1646, 2006.
- [174] D. Sander et al. Lasagne: First release. <http://dx.doi.org/10.5281/zenodo.27878>, 2015.
- [175] G. Sapiro. *Geometric partial differential equations and image analysis*. Cambridge university press, 2006.
- [176] G. Sapiro, R. Kimmel, D. Shaked, B. B. Kimia, and A. M. Bruckstein. Implementing continuous-scale morphology via curve evolution. *Pattern recognition*, 26(9):1363–1372, 1993.
- [177] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [178] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [179] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *Proc. ICLR*, 2014.
- [180] J. Serra. *Echantillonnage et estimation des phénomènes de transition minier*. PhD thesis, PhD thesis, University of Nancy, France, 1967.
- [181] J. Serra. The “false colour” problem. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 13–23. Springer, 2009.
- [182] L. Sharan, R. Rosenholtz, and E. Adelson. Material perception: What can you see in a brief glance? *Journal of Vision*, 9(8):784–784, 2009.

- [183] R. Shi, K. N. Ngan, S. Li, R. Paramesran, and H. Li. Visual quality evaluation of image object segmentation: subjective assessment and objective measure. *IEEE T-IP*, 24(12):5033–5045, 2015.
- [184] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [185] R. Socher and L. Fei-Fei. Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora. In *Proc. CVPR*, 2010.
- [186] R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.
- [187] J. Söderberg and E. Kakogianni. Automatic tag generation for photos using contextual information and description logics. In *Content-Based Multimedia Indexing (CBMI), 2010 International Workshop on*, pages 1–7. IEEE, 2010.
- [188] G. N. Srinivasan and G. Shobha. Statistical texture analysis. In *Proceedings of world academy of science, engineering and technology*, volume 36, pages 1264–1269, 2008.
- [189] 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*, 15(1):1929–1958, 2014.
- [190] C. Szegedy et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [191] Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the jvm. <http://deeplearning4j.org>, 2017. Accessed: 23.11.2017.
- [192] F. H. C. Tivive and A. Bouzerdoum. Texture classification using convolutional neural networks. In *TENCON 2006. 2006 IEEE Region 10 Conference*, pages 1–4. IEEE, 2006.
- [193] S. Townley et al. Existence and learning of oscillations in recurrent neural networks. *IEEE Transactions on Neural Networks*, 11(1):205–214, 2000.
- [194] M. Tuceryan and A. K. Jain. Texture segmentation using voronoi polygons. *IEEE transactions on pattern analysis and machine intelligence*, 12(2):211–216, 1990.
- [195] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.
- [196] J. J. van de Gronde and J. B. T. M. Roerdink. Group-invariant frames for colour morphology. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 267–278. Springer, 2013.

- [197] L. J. van Vliet, I. T. Young, and G. L. Beekers. A nonlinear laplace operator as edge detector in noisy images. *Computer vision, graphics, and image processing*, 45(2):167–195, 1989.
- [198] R. Vedantam, C. L. Zitnick, and D. Parikh. CIDEr: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015.
- [199] V. Vezhnevets and V. Konouchine. Growcut: Interactive multi-label nd image segmentation by cellular automata. In *proc. of Graphicon*, volume 1, pages 150–156, 2005.
- [200] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proc. CVPR*, 2015.
- [201] A. L. Wang, C. Y. Hu, X. M. Liu, Y. J. Iwahori, and R. Kang. A modified non-maximum suppression algorithm. In *Information Science and Electronic Engineering: Proceedings of the 3rd International Conference of Electronic Engineering and Information Science (ICEEIS 2016), January 4-5, 2016, Harbin, China*, page 69. CRC Press, 2016.
- [202] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. ICML*, 2015.
- [203] N. Xu, B. Price, S. Cohen, J. Yang, and T. Huang. Deep interactive object selection. In *Proc. CVPR*, 2016.
- [204] Y. Xu, S. Huang, H. Ji, and C. Fermüller. Scale-space texture description on sift-like textons. *Computer Vision and Image Understanding*, 116(9):999–1013, 2012.
- [205] S. Yagcioglu, E. Erdem, A. Erdem, and R. Cakici. A distributed representation based query expansion approach for image captioning. In *ACL (2)*, pages 106–111, 2015.
- [206] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo. Image captioning with semantic attention. In *Proc. CVPR*, 2016.
- [207] P. Zarchan. *Progress In Astronautics and Aeronautics: Fundamentals of Kalman Filtering: A Practical Approach*, volume 208. Aiaa, 2005.
- [208] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.
- [209] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
- [210] H. Zhang, J. Xue, and K. Dana. Deep TEN: Texture encoding network. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

- [211] H. Zhang, Z. Yi, and L. Zhang. Continuous attractors of a class of recurrent neural networks. *Computers & Mathematics with Applications*, 56(12):3130–3137, 2008.