



UNIVERSITAT POLITÈCNICA
DE CATALUNYA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
TEORIA DEL SENYAL I COMUNICACIONS

This thesis is submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy (PhD)

COMPUTER VISION BEYOND THE VISIBLE: IMAGE UNDERSTANDING THROUGH LANGUAGE

by AMAIA SALVADOR AGUILERA

Advisor: Xavier Giró-i-Nieto
Co-advisor: Ferran Marqués Acosta
Barcelona, May 2019

Abstract

In the past decade, deep neural networks have revolutionized computer vision. High performing deep neural architectures trained for visual recognition tasks have pushed the field towards methods relying on learned image representations instead of hand-crafted ones, in the seek of designing end-to-end learning methods to solve challenging tasks, ranging from long-lasting ones such as image classification to newly emerging tasks like image captioning.

As this thesis is framed in the context of the rapid evolution of computer vision, we present contributions that are aligned with three major changes in paradigm that the field has recently experienced, namely 1) the power of re-utilizing deep features from pre-trained neural networks for different tasks, 2) the advantage of formulating problems with end-to-end solutions given enough training data, and 3) the growing interest of describing visual data with natural language rather than pre-defined categorical label spaces, which can in turn enable visual understanding beyond scene recognition.

The first part of the thesis is dedicated to the problem of visual instance search, where we particularly focus on obtaining meaningful and discriminative image representations which allow efficient and effective retrieval of similar images given a visual query. Contributions in this part of the thesis involve the construction of sparse Bag-of-Words image representations from convolutional features from a pre-trained image classification neural network, and an analysis of the advantages of fine-tuning a pre-trained object detection network using query images as training data.

The second part of the thesis presents contributions to the problem of image-to-set prediction, understood as the task of predicting a variable-sized collection of unordered elements for an input image. We conduct a thorough analysis of current methods for multi-label image classification, which are able to solve the task in an end-to-end manner by simultaneously estimating both the label distribution and the set cardinality. Further, we extend the analysis of set prediction methods to semantic instance segmentation, and present an end-to-end recurrent model that is able to predict sets of objects (binary masks and categorical labels) in a sequential manner.

Finally, the third part of the dissertation takes insights learned in the previous two parts in order to present deep learning solutions to connect images with natural language in the context of cooking recipes and food images. First, we propose a retrieval-based solution in which the written recipe and the image are encoded into compact representations that allow the retrieval of one given the other. Second, as an alternative to the retrieval approach, we propose a generative model to predict recipes directly from food images, which first predicts ingredients as sets and subsequently generates the rest of the recipe one word at a time by conditioning both on the image and the predicted ingredients.

Acknowledgments

This thesis has come into being thanks to the contributions and the support of many people, who I can only but thank for accompanying me along this journey.

First, I must thank my advisor Xavier Giró for all these years of unconditional support, and for believing in me from the very first day. I am thankful to him for peaking my curiosity for computer vision almost seven years ago now, and for his patience in teaching me how to do research since then. I am grateful to him for his contagious enthusiasm and for always seeing the bright side, no matter what. I sincerely want to thank him for encouraging me to do things and go places I could have never imagined. I am certain that in many ways, these experiences have shaped me into the person I am today.

I would also like to express my most sincere gratitude to my co-advisor Ferran Marqués, for his wisdom and for the consistently valuable advice he has given me along the years.

I wish to broadly acknowledge all the colleagues that I have crossed paths with at UPC—from the ones who have come and gone to the ones that remain here still, thank you for making work enjoyable even in the most nerve-racking moments. A special thanks goes to Santi, for squeezing my brain with your crazy thoughts and for sharing your ideas with no filters. To Míriam and Víctor, thank you for showing up one day and staying—you made the bumpy road a little easier to handle. And to Míriam in particular, thank you for getting me so well, and for our countless never ending chatters I can't seem to get enough of. Finally, to Eva: I am thankful that I got to go through this adventure by your side, from the early days when we clearly didn't know what we were doing, until today. I hereby follow your steps.

I acknowledge that this thesis would not have been possible without the financial assistance of the Image Processing Group (GPI) at UPC and the Spanish Government through the FPI grant and the projects TEC2013-43935-R and TEC2016-75976-R. I must also acknowledge Albert Gil and Josep Pujal, for setting up and maintaining the computing server at GPI, and also for the high-quality technical support they steadily provide us with. It is thanks to them that we get to only think about research.

I would like to extend thanks to colleagues, advisors and friends that I met and learned from during my internships at INP-ENSEEIH, Insight-DCU, NII, MIT-CSAIL and FAIR. I must specially thank Vincent Charvillat, Axel Carlier, Oge Marques, Kevin McGuinness and Noel O'Connor, for showing me the wonders of collaborative research before I even started the Phd. I sincerely want to thank Shin'ichi Satoh and Antonio Torralba, for welcoming me to join their labs for a few months and providing me with wonderful learning experiences. Finally, my deepest gratitude goes to Adriana Romero and Michal Drozdal for sharing their knowledge with me, and most importantly for guiding me on the right path when I most needed it.

I also want to thank my friends, who make me enjoy life at its finest and whose support has kept me afloat through hard times. Thanks to Helena, for always listening to all my monologues, but mostly for seeing the world as I do. To Mireia, thank you for always being there when I need you, and for your inexplicable ability to still make me have fun, laugh and even sometimes behave like a teenager. A big thanks goes to Antonio, Ricard and Dani, for all the trips together and those that will come. I also wish to thank Guillem, Sergi, Xavi, Tamara, Martí, Mireia, Marina and Quique, for making me feel at home when I'm around you.

Last but not least, I would like to thank my family, for putting up with me and supporting me in every sense of the word. I want to thank my brother Emili, for all the silliness and the five-year-old jokes that luckily nobody else understands. To my father Emilio, thank you for silently looking out for me, and for always having the solution to all my problems. Finally, I thank my mother Josefa, for being the source of all the inspiration I ever needed in life, and for teaching me everything that matters.

Contents

0.1	Motivation	2
0.2	Deep Learning	6
I	Visual Instance Search	17
1	Introduction	19
1.1	Content-based image retrieval	19
1.2	Instance Search	20
1.3	Related Work	21
1.4	Datasets	23
1.5	Metrics	23
2	Bags of Deep Visual Words	25
2.1	Bags of Deep Local Features	26
2.2	Image Retrieval Pipeline	27
2.3	Experiments	28
2.4	Conclusion	34
3	Object Detectors for Instance Search	35
3.1	ConvNets for Object Detection	36
3.2	Deep Representations for Images and Regions	37
3.3	Fine-tuning Faster R-CNN	38
3.4	Image Retrieval Pipeline	38
3.5	Experiments	39
3.6	Conclusion	45
	Summary	47
II	Image-to-Set Prediction	49
4	Introduction	51
5	Multi-label Image Classification	53
5.1	Related Work	54
5.2	Image-to-Set Prediction Methods	55
5.3	Experiments	59
5.4	Conclusion	64
6	Recurrent Instance Segmentation	65
6.1	Related Work	66

6.2	Model	67
6.3	Experiments	70
6.4	Conclusion	78
	Summary	79
	III Image-to-Recipe Prediction	81
7	Introduction	83
7.1	Food Understanding	84
7.2	Recipe1M Dataset	85
7.3	Language Modeling	86
7.4	Text Representations	88
7.5	Language and Vision	89
8	Recipe Retrieval	93
8.1	Methodology	93
8.2	Experiments	97
8.3	Conclusion	103
9	Recipe Generation	105
9.1	Methodology	106
9.2	Experiments	109
9.3	Conclusion	115
	Summary	117
	Conclusions	119
	Publications	121
	Bibliography	123

Introduction

Vision (or visual perception) is the process of discovering from images what is present in the world, and where it is located [132]. As humans, we are able to seamlessly understand what we see: we identify objects and their relationships, which allows us to infer knowledge from what we perceive and interact with. Vision has been studied for centuries [148] with the goal of understanding how the visual system works and even mimic its behavior with computer programs.

Computer vision is the scientific field that studies the theory behind artificial systems that are able to extract relevant information from images. The foundations of computer vision started in 1963, with the first attempt to extract edge-like 3D structures from 2D views of polyhedrons [171]. In 1966, researchers from MIT wrote a project proposal [154] where the goal was to construct an artificial visual system that could identify objects in images. This project was intended to be solved over the course of a single summer. Their optimism can perhaps be explained by the numerous computer programs involving logical and algebraic operations that succeeded at the time. Paradoxically, it turned out that the most difficult human skills to reverse engineer are those that we perform unconsciously [142]. What started as a short-term project in the 1960s later evolved into an entire research field that has been studied for decades and, despite great progress has been made, it still remains unsolved.

Early research in computer vision in the 1970s focused on extracting 3D geometrical information from images with the purpose of understanding their contents. Low-level computer vision algorithms such as edge and corner detection were extensively studied in that decade. One of the revolutionary ideas in computer vision was the bottom-up representation of images [132], in which lower-level image representations served as intermediate steps to compose a 3D model representation of the scene. Insights from [132] shifted the field of computer vision towards hierarchical image representations in the 1980s [126, 23]. Since the 1990s, the computer vision field significantly moved towards the design of algorithms that aimed at recognizing objects directly from 2D image representations, thus bypassing the previously mandatory step of obtaining 3D object models. Instead, objects and their parts were encoded with engineered representations extracted from still images [202, 127, 44, 58]. In the 2000s, the emergence of curated image datasets [54, 177] allowed the benchmarking of computer vision algorithms.

Computer vision witnessed a breakthrough in 2012, when Krizhevsky et al. [102] won the ImageNet classification benchmark with a deep neural network trained on a GPU. The neural network, which is commonly referred to as AlexNet, reduced the top-5 error from 26% to 15.3%, a substantial improvement with respect to previous methods based on hand-crafted features. Although neural networks had already been applied to image classification tasks [106], AlexNet is known to be one the most influential works in com-

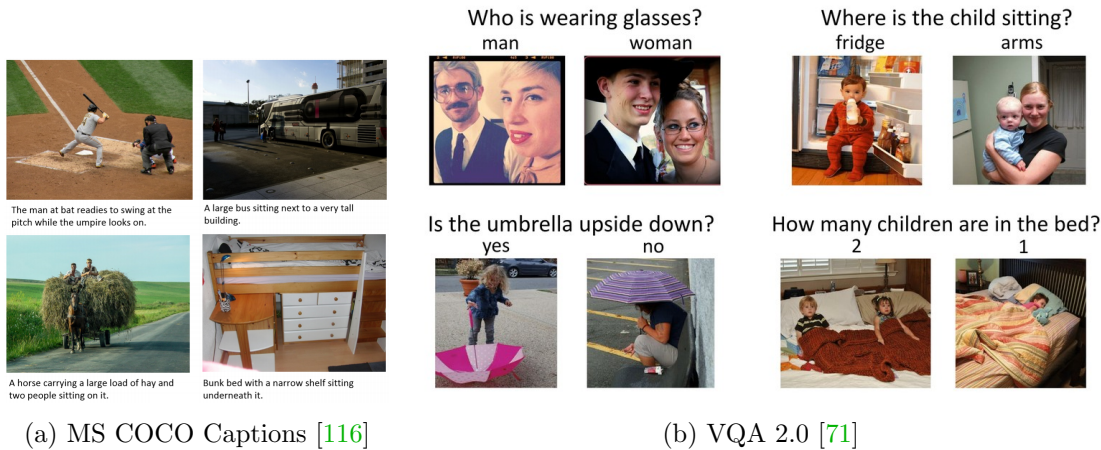


Figure 0.1: Samples from (a) MS COCO Captions and (b) VQA 2.0 datasets. Figures taken from [35] and [71], respectively.

puter vision, which led to a change in paradigm to move away from hand-crafted image representations [127, 202] towards learned ones. Since then, deep learning algorithms have vastly dominated the computer vision field, allowing astonishing improvements in traditional applications such as image categorization [182, 81], image detection [64, 167], or image segmentation [125, 79], but also opened the doors to new challenges such as text-conditioned image generation [20, 224] image captioning [201, 215] or visual dialog systems [45]. These new challenges have brought computer vision closer to research communities studying other data modalities, such as natural language or audio and speech signal processing, in the seek of intelligent systems that can simultaneously reason about different modalities and infer knowledge from them.

Motivation

The concept of *modality* refers to the type of representation in which a certain information is encoded, i.e. the way in which something occurs or is experienced. The human experience of the world is inherently multimodal. As humans, we learn concepts by naturally aligning the cues that we perceive, which allows us to use multiple sensors to extract the same unit of information (e.g., we can tell a storm is coming from both a dark sky and the sound of thunder strikes). Although we learn from exploring the world on our own, we also acquire knowledge through communication with others. Human language allows us to describe complex ideas, events or things that we observe in the world to others. Humans can also imagine unseen concepts when described to them, and describe what they see with words at different degrees of detail. While human language can be expressed in multiple modalities (e.g., spoken language uses the auditive modality, sign languages and writing use the visual modality, braille writing uses the tactile modality), for the purposes of this thesis we refer to language in its written modality (text).

In the past few decades, computer vision has largely focused on designing algorithms to recognize what is visible in images (i.e. naming and locating object entities in images), being image classification [102, 182, 177, 81], object detection [64, 63, 167] or object segmentation [125, 79] some of the tasks that have received most of the attention in the research community. However, there is a lot more to a picture than meets the eye.

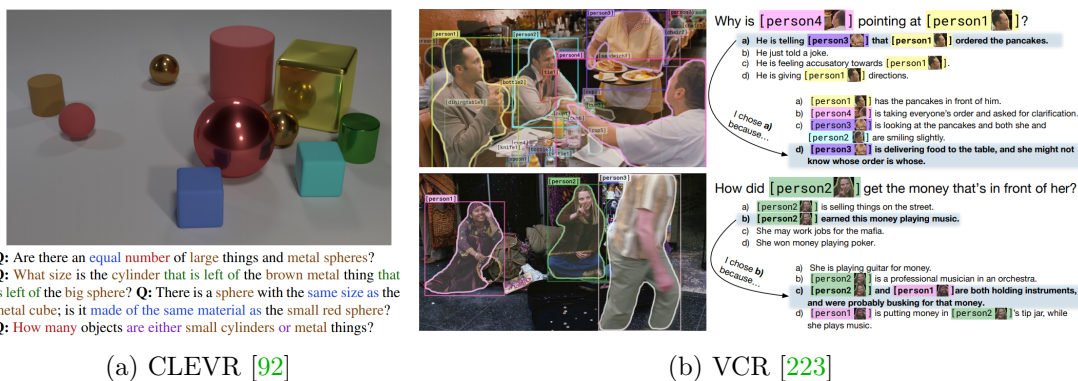


Figure 0.2: Samples from (a) CLEVR and (b) VCR datasets. Figures taken from [92] and [223], respectively.

Despite the great advances in fundamental tasks such as single-label image classification to more challenging tasks such as semantic instance segmentation [79], while current state of the art algorithms have been demonstrated to be useful for practical applications, they only begin to scratch the surface of all the knowledge that can be derived from an image.

Achieving true visual understanding of complex scenes goes far beyond object recognition. In this direction, the computer vision community recently witnessed the emergence of new datasets and benchmarks to accommodate research in the intersection of vision and language [116, 71]. Since then, tasks such as image captioning [201, 215, 128] and visual question answering [129, 180] have gained attention in the community. While these tasks require some degree of higher level reasoning in order to be solved, most image captioning or visual question answering benchmarks are composed of samples in which textual information is still heavily tied to the visual appearance of objects. Most captions in the MS COCO dataset [116] simply enumerate the objects and describe their surroundings, and many of the questions in the VQA 2.0 dataset [71] are related to object appearance (see Figure 0.1 for examples). Thus, the performance of such methods is still broadly bounded to the ability of current image recognition systems to name, locate and count objects in images. Only recently, a few datasets requiring a higher degree of reasoning have been released. Visual reasoning has been studied in controlled rendered scenarios with the CLEVR dataset [92], which challenges computer vision algorithms to answer questions requiring counting, comparing and logical reasoning capabilities. One step further is the newly released Visual Commonsense Reasoning (VCR) dataset [223], including questions related to real world objects and their relationships, which also require higher level understanding of the scene in order to be answered. Figure 0.2 shows examples of samples included in the CLEVR and VCR datasets.

The appearance of large scale datasets linking images with language [116, 3, 71, 223] aided the development of methods that can generate sentences describing images [201, 128], or answer questions about them [129]. Although these datasets introduce many intrinsic challenges that push the boundaries of visual learning methods, these are constrained to model interactions between language and vision involving short sentences (e.g., the average caption in MS COCO contains 11.3 words). While recent works have attempted at generating longer paragraphs (> 60 words) describing image contents [101], we argue that language can be used for a lot more than just describing what is visible.

Visual and textual data naturally co-exist in the web: articles are often supported with visual elements in digital newspapers, and social networks enable human interactions about visual data in forum threads or comment sections. In these cases, the language may not be necessarily describing the visual contents (e.g., opinions, jokes or topic discussions often emerge from or are supported with images). Notably, all the aforementioned datasets including paired image and textual data are either artificially generated [92] or require the collection of human annotations [116, 3, 71, 223], which can be expensive to obtain and are constrained by the limitations of the annotation tool and the annotator. In contrast to human-curated datasets, web-based datasets [38, 151, 179] can be obtained by crawling online resources, which make them cheaper to collect and can be more representative of real world interactions of vision and language in different domains.

In this context, the research presented in this thesis closely follows and is influenced by the evolution trends of the computer vision field over the past decade. The first two parts of the thesis largely focus on both extracting and optimizing representations from pre-trained neural networks for visual recognition tasks, which ultimately lead to the development of methodologies that connect vision with language in the domain of food images and cooking recipes.

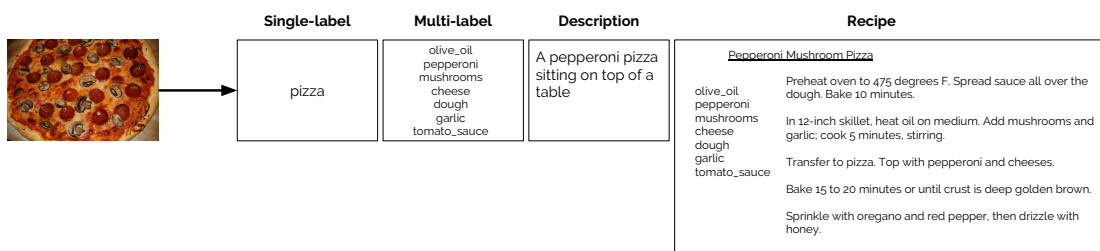


Figure 0.3: Examples of language-based outputs for an input image.

The vast availability of online recipe collections along with example images presents the possibility of training machines to link images and language directly from raw data. Food data also offers new perspectives on topical challenges in computer vision like finding representations that are robust to occlusion and deformation (as occur during ingredient processing). As entities, images and language are connected with many-to-many relationships. Just as there are many sentences that can be used to describe the same image, different images can be described using the same words. In the context of food images and cooking recipes, visual ambiguities caused by ingredient processing during cooking frame the task of intuiting recipes from images as one that seeks to find a *plausible* recipe rather than the true one (i.e. the one that was originally followed to obtain the cooked dish in the image). Inferring a recipe from a picture is a task that pushes machine algorithms beyond recognizing image contents and requires visual reasoning to infer probable recipes that are coherent with image contents.

This dissertation is framed in the intersection of vision and language for cross-modal learning, i.e. designing representations that extract common knowledge from the two aforementioned modalities, which allow the recovery of one given the other. Cross-modal learning is not to be confused with multi-modal learning, which involves processing and relating information from multiple modalities for a common task (e.g., video classification with audiovisual features). Figure 0.3 shows examples of possible language-based information that can be obtained from an image. One can provide a single word that

represents its broad semantics (single-label classification), a set of labels that are present in the image (multi-label classification), a broad description of the image (image captioning), or long structured text (e.g., a paragraph, a story or a cooking recipe).

The goal of this thesis is to design solutions that accomplish the latter, particularly in the context of food data, i.e. learning cross-modal representations that allow obtaining cooking recipes directly from food images. This thesis tackles the image-to-recipe problem with two different strategies. The first one considers a retrieval solution, where the challenge lies in obtaining compact representation of images and recipes that enable the retrieval of one given the other. The second one takes a generation-based approach and aims to predict a cooking recipe (the title, its ingredients and the cooking instructions) directly from a food image.

Cooking recipes are structured long textual documents which are non-trivial to encode, model or generate. For this reason, the first two parts of this thesis take intermediate steps in both directions (i.e. retrieval- and generation-based approaches), which then lead to the proposed solutions to the practical image-to-recipe application. All the contributions of this thesis have their foundations on deep neural networks. For the sake of completion, the technical background on deep neural networks is provided in Section 0.2. The expert reader can choose to skip this section.

Part I Image Retrieval	Part II Image-to-Set Prediction	Part III Image-to-Recipe Prediction
Chapter 2 Off-the-shelf representations	Chapter 5 Multi-Label Image Classification	Chapter 8 Recipe Retrieval
Chapter 3 Query-optimized representations	Chapter 6 Instance Segmentation	Chapter 9 Recipe Generation

Table 0.1: Thesis structure.

Table 0.1 illustrates the structure of the remainder of this thesis. First, Part I is dedicated to explore the suitability of using features extracted from hidden layers of deep neural networks as image representations for content-based image retrieval. Both off-the-shelf and query-specific deep representations are explored in Chapters 2 and 3, respectively. Insights learned in this part of the thesis will be the basis of the retrieval-based image-to-recipe approach (introduced in Chapter 8). Analogously, generating recipes directly from images requires systems that predict textual documents as structured sentences, which have been largely explored in the literature [190, 196, 128]. However, recipe generation also requires estimating ingredients, which may naturally be represented as a sets (i.e. a variable-length collections of unique and interrelated constituents). With this purpose, Part II of the thesis is dedicated to set prediction using deep neural networks, exploring two different tasks, namely multi-label classification (Chapter 5) and semantic instance segmentation (Chapter 6). Finally, transitional steps taken in Parts I and II lead to Part III of this thesis, where the design and comparison of the two aforementioned retrieval-based (Chapter 8) and generation-based (Chapter 9) image-to-recipe approaches are presented.

Deep Learning

In the field of artificial intelligence, the study of algorithms that are capable of performing a specific task by relying on patterns from data instead of hard-coded instructions is known as machine learning. Machine learning enabled computers to tackle problems which require the extraction of knowledge from the real world to make decisions given raw data. Although there is a vast family of models in machine learning, such as support vector machines, decision trees and bayesian networks, the best performing machine learning solutions to many applications in vision, speech and natural language processing are nowadays based on deep neural networks.

This section briefly introduces the necessary background on deep neural networks, including common architectures that will be used in the three main parts of this thesis.

The Perceptron

The basic unit in a deep neural network is the neuron, also called the perceptron [174]. A perceptron takes several inputs $\{x_1, \dots, x_n\}$ and produces a single output o_j . Figure 0.4(a) provides an illustration of the perceptron model. The perceptron associates a weight w_i to each input, which together constitute the parameters of the function. The output of the neuron is given by a linear combination of weighted inputs followed by a non-linear activation function φ :

$$o_j = \varphi \left(\sum_{i=1}^n w_i \cdot x_i + b_j \right), \quad (1)$$

where b_j denotes the bias term for the neuron j , which is a separate weight applied to a constant x_0 input permanently set to 1. The variables w_i and b_j constitute the parameters that define the behavior of the neuron. The bias term will be dropped from future equations for simplicity.

The activation function φ usually takes the form of a sigmoid, hyperbolic tangent (tanh) or rectified Linear unit (ReLU) [145], which are compared in Figure 0.4(b). In a binary classification problem, the output of the neuron after a sigmoid activation $o_j \in \{0, 1\}$ can be interpreted as a linear decision boundary classifier, which can be thresholded to obtain the final decision y_j :

$$y_j = \begin{cases} 1 & \text{if } o_j \geq th_j \\ 0 & \text{if } o_j < th_j \end{cases} \quad (2)$$

Multi-class classification problems can be handled with multiple neurons operating on the same input. In this scenario, it is common to use the softmax non-linearity to obtain a probability distribution over C classes:

$$o_j = \frac{e^{net_j}}{\sum_{j=1}^C e^{net_j}}, \quad (3)$$

where net_j is the output of the j^{th} neuron before the activation function.

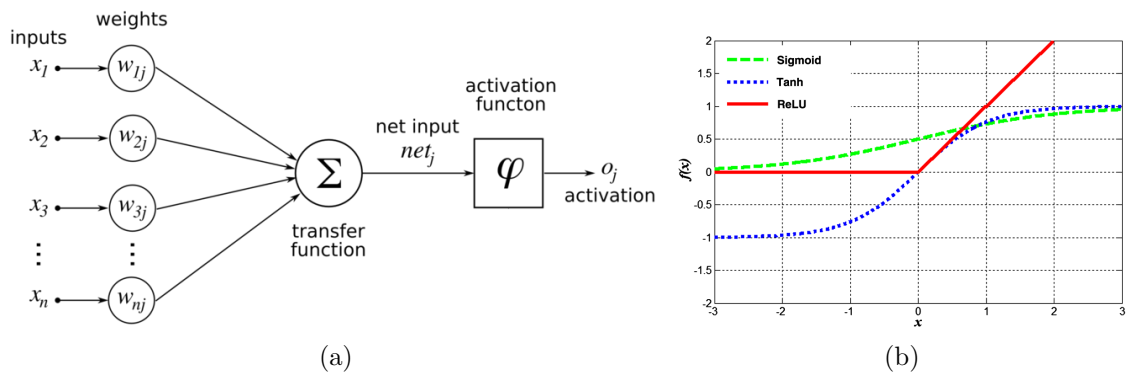


Figure 0.4: 0.4(a): The Artificial Neuron (Source: Chrislb, Wikipedia). 0.4(b): Examples of activation functions (taken from [204]).

In this case, the output $O = \{o_1, \dots, o_C\}$ is calculated by several neurons that operate on the same input. Arrays of neurons operating on the same input are commonly referred to as *layers*. In the multi-class classification problem described in this section, the network is composed of two layers: the input layer (composed of values x) and the output layer (composed of weights w).

Neural Networks

While linear classifiers are suitable when the input data is linearly separable, real world data (e.g., image pixels), very rarely falls into this category, and instead requires more complex functions that can tackle high-level decisions (e.g., object classification) from raw inputs.

Neural networks are based on a collection of connected neurons, which are arranged in layers. They are composed of an input layer and an output layer, which are connected by a variable number of hidden layers. Neural networks can approximate complicated functions mapping inputs to outputs by decomposing them into a series of simple transformations learned in the parameters of each hidden layer. Increasing the number of layers in a neural network is also referred to as increasing its *depth*, which is the reason we conventionally refer to these models as *deep neural networks*, and we understand *deep learning* as the field that studies the design and optimization of deep neural networks.

The arrangement of neurons in multiple layers leads to learned hierarchical representations of the input data, which is transformed in each layer with the purpose of becoming linearly separable at the output layer. Once trained, these networks have been demonstrated to learn data representations at different levels of abstraction (from edges and corners to object parts) in each of their hidden layers [222].

Given a set of M training samples consisting of input and label pairs $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=0}^M$, a deep neural network can estimate a mapping from inputs x to outputs y by adjusting its weights θ (weights w_i and biases b_j) according to a loss function $\mathcal{L}(\hat{y}, y)$ using an optimization algorithm, such as gradient descent. Given the loss function associated to the outputs of a neural network, gradient descent updates the weights in the opposite

direction of the gradient:

$$\theta_{i+1} = \theta_i - \lambda \nabla \mathcal{L}(\hat{y}, y), \quad (4)$$

where λ is the learning rate, which determines the magnitude of the step taken. Other popular optimization algorithms that are often used instead of gradient descent are Adam [99] or RMSProp [72]. The weights of any intermediate layer in a deep neural network can be updated by using the chain rule to propagate the gradient of the loss back to the first layer (known as the backpropagation algorithm [176]). In practice, obtaining the gradients for a loss function computed on all training samples M is computationally intractable for large-scale datasets. For this reason, mini-batch Stochastic Gradient Descent [170] is used instead, where the gradient is estimated on a batch of N training samples. The process of updating the weights of the model so that it gets better at a particular task, whose performance is evaluated with a loss function $\mathcal{L}(\hat{y}, y)$, is what we understand as *learning*.

Feed-forward Neural Networks

The simplest neural network is the multi-layer perceptron (MLP), in which each layer is composed of several neurons, each of them connected to all neurons in the previous layer. Layers that fulfill this property are commonly called *fully connected* layers. Given the output of its previous layer in a neural network: $h_{k-1} \in \mathbb{R}^{D_{k-1}}$ (where D_{k-1} denotes the number of neurons in layer $i - 1$), the output of the the next fully connected layer $h_k \in \mathbb{R}^{D_k}$, in matrix notation, is given by:

$$h_i = W_i^T \cdot h_{k-1}, \quad (5)$$

where $W_k \in \mathbb{R}^{D_{k-1} \times D_k}$ are the weights of the k^{th} fully connected layer.

As more hidden layers are added to an MLP, the amount of weights of the neural network increases very quickly, which leads to high-capacity models that exhibit poor generalization capabilities. Further, fully connected layers do not exploit the structure of the input data in any way (all inputs are connected to all outputs), which may not be the optimal approach when dealing with structured data such as images.

Convolutional Neural Networks

Convolutional Neural Networks, also known as Convolutional Networks, ConvNets, or CNNs, are a particular kind of neural network especially suitable for processing data with grid-like structure, such as images (an image is commonly represented as a 2-D grid of pixel values). ConvNets can be understood as neural networks that use the convolution operation in place of general matrix multiplication in their layers.

Given a one-dimensional input $x = \{x_1, \dots, x_d\}$, we can define a convolutional kernel with an array of weights W_k of size K , which can be applied to the input x with the following

equation:

$$h_k(i) = (W_k * x)(i) = \sum_{m=1}^K x(i-m) \cdot W_k(m), \quad (6)$$

where $*$ denotes the convolution operation.

In the case of two-dimensional inputs (e.g., a grayscale image $I \in \mathbb{R}^{H \times W}$), a convolutional kernel defined by a matrix of weights W_k of size $K_h \times K_w$ is applied as:

$$h_k(i, j) = (W_k * I)(i, j) = \sum_{m=1}^{K_h} \sum_{n=1}^{K_w} I(i-m, j-n) \cdot W_k(m, n) \quad (7)$$

The convolutional layer is the generalization of applying multiple convolutional kernels to an input of arbitrary dimensionality. As an example, a color image can be expressed as $I \in \mathbb{R}^{H \times W \times C}$, where C denotes the dimensionality, which equals to 3 for RGB images. A convolutional layer with multiple kernels that can operate on top of image I is defined by a matrix of weights W_k of size $K_h \times K_w \times C \times D_k$, and is applied to image I with the following equation:

$$h_k(i, j, d) = (W_k * I)(i, j, c) = \sum_{m=1}^{K_h} \sum_{n=1}^{K_w} \sum_{c=1}^C I(i-m, j-n, c) \cdot W_k(m, n, c, d) \quad (8)$$

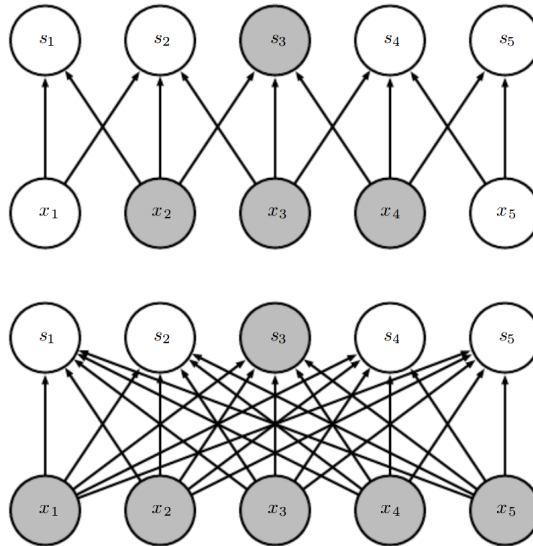


Figure 0.5: Sparse vs Dense Connectivity. Figure from [69]. Neuron s_3 is only affected by inputs x_{2-4} in a convolutional layer (top), while it is affected by all inputs in a fully-connected layer (bottom).

Since the kernel size is usually set to be smaller than the input size, replacing matrix multiplication operations with convolutions transforms fully connected layers into sparsely

connected ones, where neurons in an arbitrary layer are only connected to a few neurons in its preceding layer. Figure 0.5 illustrates this property. Sparse connectivity also affects the receptive field of a neuron, which is defined as the portion of the input that is visible to each neuron in the network. While the receptive field of neurons in fully-connected layers corresponds to the full input, neurons in convolutional layers have a limited receptive field, which increases with layer depth.

Further, convolutional layers use the same kernel weights at different locations in the input. This can be observed in Equation 8, where the same kernel weight $W_k(m, n)$ is used to obtain the output of the convolutional layer in all positions (i, j) . Intuitively, this is a desirable property to have when dealing with images as inputs, since a learned feature extractor (e.g., an edge detector) can be useful in all image locations.

ConvNets have been successfully used in applications in computer vision. The first deep convolutional neural network trained on the large-scale ImageNet dataset was AlexNet, consisting of seven convolutional hidden layers (see Figure 0.6). Convolutional layers are usually applied on zero-padded inputs in order to maintain the spatial dimension of the input features. However, training a deep ConvNet on images of size $224 \times 224 \times 3$ would require large amounts of memory. For this reason, max-pooling operations are commonly applied between convolutional layers of deep ConvNets, which reduce the spatial dimensionality of the convolutional feature maps as depth increases. Max-pooling operations also increase the receptive field of neurons in subsequent layers.

Subsequent works proposed deeper architectures such as VGG [182] or ResNet [81], which outperformed AlexNet by using more convolutional layers with smaller kernels [182] and incorporating residual connections [81]. While originally trained for the ImageNet classification benchmark, these models have been widely used as feature extractors for other image classification tasks, as well as for different computer vision applications (e.g., object detection or segmentation).

The process of solving a task A by adapting or reusing a model that has been trained for a task B is called *transfer learning*. Usually, transfer learning is conducted by *fine tuning* a model, which is understood as the process of slightly modifying the weights of a previously trained model, so that it can solve a different task. Depending on the amount of available data, one can choose to change the weights of only a few or many layers of the network.

Pre-trained deep neural networks are also used as feature extractors to encode images into descriptive representations useful for other tasks whose outputs cannot directly be modeled with a neural network (e.g., image retrieval). Altogether, transfer learning strategies make it possible to use deep neural networks for applications for which large amounts of data are not available for training. All methods presented in this thesis use transfer learning, as they are built on top of neural architectures such as VGG16 [182] or ResNet-50 [81], which have been pretrained for image classification on ImageNet [177].

Neural Networks for Structured Outputs

The previous sections have covered the basics of neural networks applied to classification, where the desired output is a single categorical label. However, current deep neural networks can be trained to perform more complex computer vision tasks, which require

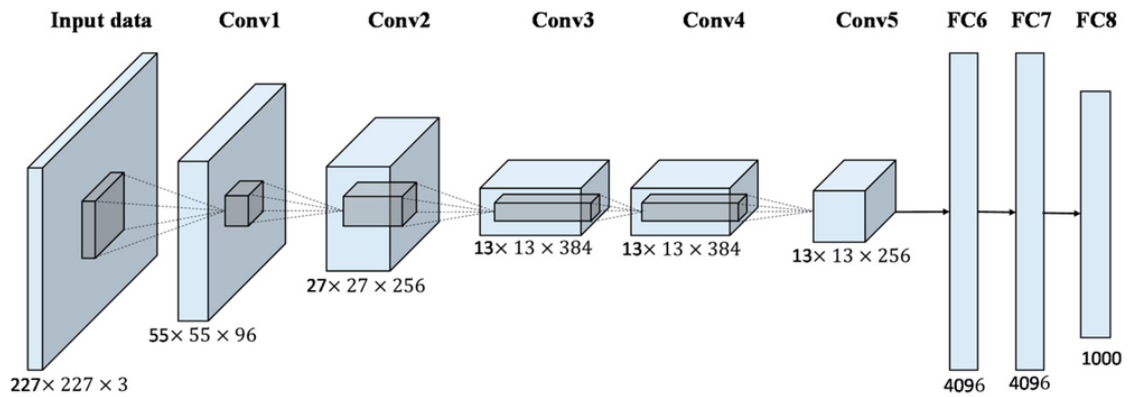


Figure 0.6: The AlexNet architecture. Figure from [75].

output structures that go beyond one-dimensional vectors.

Structured output tasks involve the prediction of tensors containing multiple values which are related to one another. Examples of computer vision applications that require structured outputs include semantic segmentation, where the goal is to provide a categorical output to every pixel in an image, or image captioning, where a textual description must be provided for an image. In semantic segmentation, decisions for neighboring pixels are heavily correlated; in image captioning, the different words in the description must form a valid sentence.

In this section, we review the particularities of two computer vision problems with structured outputs, namely dense prediction and sequence prediction. We highlight common neural network models that are used to solve each of them, which are the main building blocks in the contributions presented in this thesis.

Dense Prediction

In general terms, dense prediction refers to the process of providing an output for each element in the input. In computer vision, input elements are image pixels, which turns the dense prediction task into one that requires providing a decision (e.g., a scalar, or a categorical probability distribution) for each pixel in the image. Dense prediction in ConvNets was initially achieved by transforming fully connected layers from image classification architectures into convolutional ones by re-arranging the weights of the fully connected layer into those of a 1×1 convolutional layer. We illustrate this process with an example on the AlexNet architecture [102]. In AlexNet, the first fully-connected layer $fc6$ receives the output of the fifth convolutional layer $conv5$, which corresponds to a feature volume of size $13 \times 13 \times 5256$. Layer $fc6$ outputs a vector of $D = 4096$, which is achieved with a matrix of weights $W_{fc6} \in \mathbb{R}^{9216 \times 4096}$. Conversely, the same output can be obtained with a convolutional layer with weights $W_{conv6} \in \mathbb{R}^{6 \times 6 \times 256 \times 4096}$, which has the same number of parameters as W_{fc6} . Since the height and width of the convolutional filter are set to be exactly the same as the input volume's, the output would be of dimension $1 \times 1 \times 4096$. The same process is applied to subsequent fully connected layers in the ConvNet. After this conversion, one can feed an image of higher resolution to AlexNet, or remove pooling layers in order to obtain a bigger feature map in $conv5$. Then, the output of converted layers applied to $conv5$ would be a dense prediction.

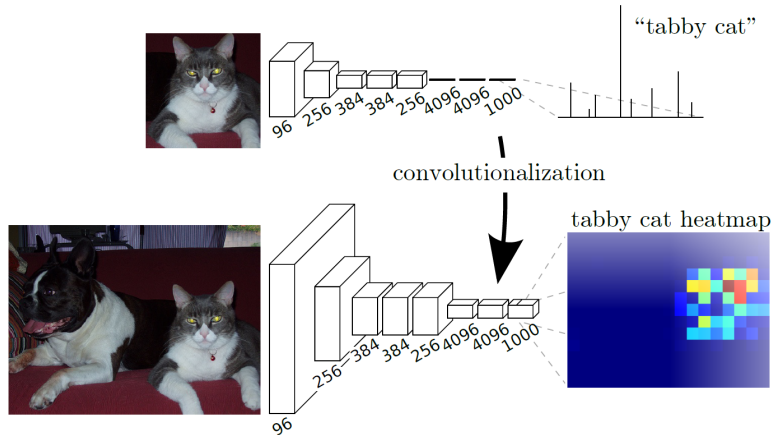


Figure 0.7: *Convolutionalization* of fully-connected layers. Figure from [125].

Figure 0.7 illustrates the *convolutionalization* process for fully-connected layers. Dense prediction architectures are often designed to make use of convolutional features from several layers to predict their output. This procedure is referred to as *skip connections* (illustrated in Figure 0.8), which allow to recover lower level features from preceding layers to refine the final prediction. The final output heatmap can be upsampled to image resolution with bilinear interpolation or through a learnable upsampling (also known as *deconvolution* or *transposed convolution*). ConvNets trained for the dense prediction task are often called *fully-convolutional neural networks*. Dense prediction architectures will be used in Chapter 6 of this thesis to predict object segments as elements of a set for semantic instance segmentation.

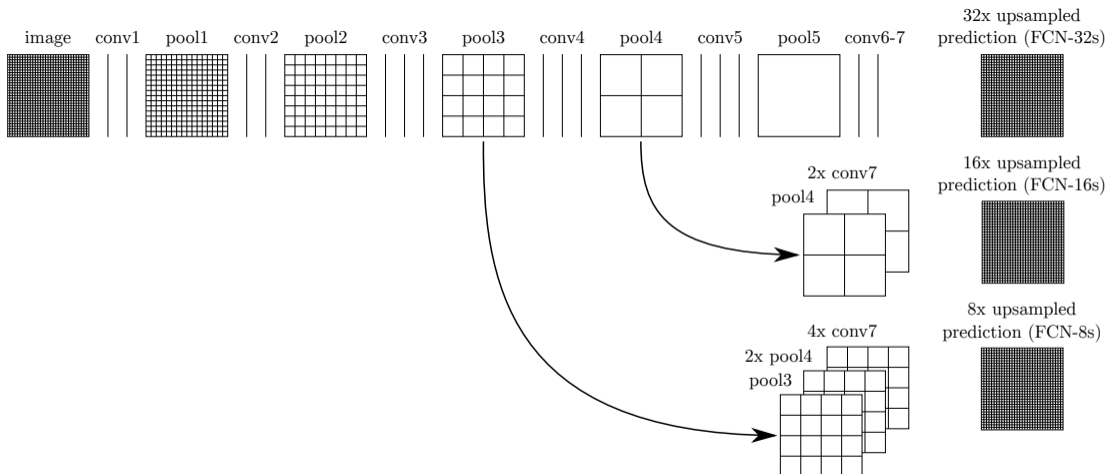


Figure 0.8: Skip connections for dense prediction. Figure from [125].

Sequence Prediction

Fully-convolutional neural networks exploit the grid-like structure from images in order to make a coherent dense predictions in their output. However, elements in structured outputs can require different relationship patterns. One important family of structured outputs are sequences. Although many problems in computer vision require dealing

with input sequences (e.g., video analytics), for the purposes of this thesis we will focus on computer vision problems in which the input is a still image, and the output is a sequence. As previously introduced, one common problem in computer vision is image captioning, where the task is to provide a textual natural language sentence for the input image. Language is inherently sequential, since valid sentences are constructed in particular word arrangements which make them meaningful. Modeling sequences with neural networks is a challenging task, since it requires to retain information about past predictions to infer subsequent ones. Models that enable such kind of feedback in their input are known as *auto-regressive* neural networks. In this section, we briefly review two kinds of auto-regressive neural networks that will be used in this thesis, namely recurrent neural networks (RNNs) and attention-based models.

RNNs [53] can be understood as fully-connected networks that take two elements as inputs: the current input at time-step t , and their own output at the previous time-step $t - 1$. Thus, the output of a vanilla RNN for time-step t is computed as:

$$h^{(t)} = g(\mathbf{U} \cdot x^{(t)} + \mathbf{W} \cdot h^{(t-1)}) = g(\mathbf{U} \cdot x^{(t)} + \mathbf{W} \cdot g(\dots(\mathbf{U} \cdot x^{(t-T)} + \mathbf{W} \cdot h^{(t-T)})\dots)), \quad (9)$$

where \mathbf{W} and \mathbf{U} are the learnable parameters of the network. The same function g with the same parameters \mathbf{U} and \mathbf{W} is applied at every time-step, which allows RNNs to operate on sequences of any arbitrary length. RNNs are optimized with gradient-based approaches using back-propagation to calculate the gradient of recurrent neurons. The back-propagation algorithm applied to an unfolded RNN is called back-propagation through time (BPTT). Figure 0.9(a) depicts the side and unfolded views of the computational graph for an RNN. Given the hidden state of an RNN at time-step t , a third matrix of weights \mathbf{V} is used to transform it into an output $o^{(t)}$, which is used to compute the loss $L^{(t)} = \mathcal{L}(y^{(t)}, o^{(t)})$, where $y^{(t)}$ is the ground truth at time-step t and \mathcal{L} is the loss function (e.g., cross-entropy loss in the case of classification). While information from previous time-steps is preserved in the hidden state, one common practice is to use the actual output $o^{(t-1)}$ from the previous time-step as an additional input to the RNN. Models that include such recurrent feedback from their outputs can be trained with *teacher forcing*, a procedure in which the output $o^{(t-1)}$ is replaced with the ground truth $y^{(t-1)}$ to be used as feedback at time-step t during training. At test time, since the correct output $y^{(t-1)}$ is not available, it is approximated with $o^{(t)}$ instead. Figure 0.9(b) illustrates the teacher forcing procedure.

One of the most critical problems with vanilla RNNs is that they are prompt to suffer from vanishing or exploding gradients when unrolled for long sequences, due to their recursive multiplicative operation in Equation 9. Variations of the vanilla RNN such as the LSTM [82] and GRU [37] have been proposed in order to mitigate this problem.

Long Short Term Memory Networks (LSTM) [82] are a kind of RNNs specially designed to better model long-term dependencies. LSTMs are composed of four functions (also called *gates*) that interact with each other in a particular way in order to obtain the t^{th} output $h^{(t)}$ given the input $x^{(t)}$ and the previous output $h^{(t-1)}$:

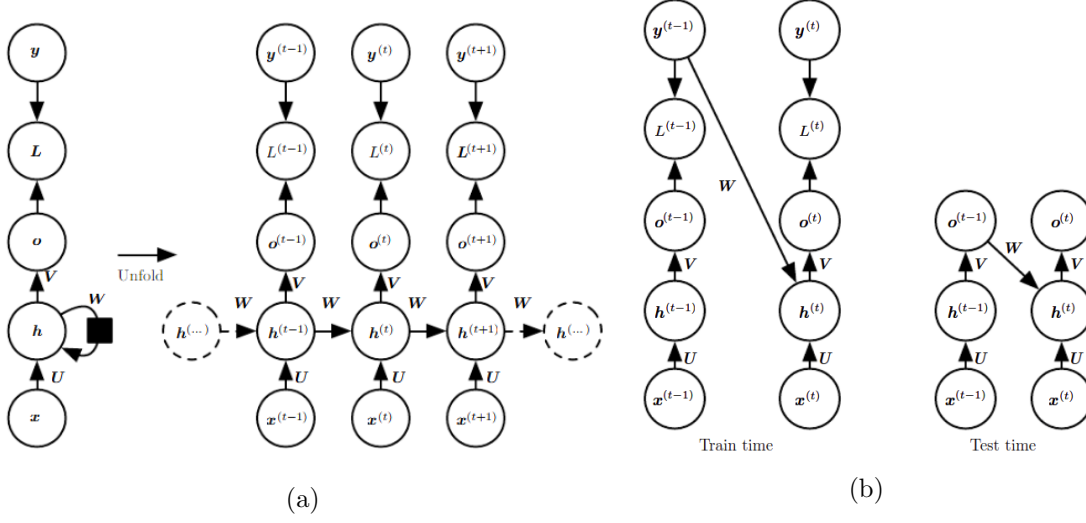


Figure 0.9: 0.9(a):Unfolding an RNN. 0.9(b): Teacher forcing. Figures from [69].

$$\begin{aligned}
 f^{(t)} &= \sigma(g(\mathbf{U}_f \cdot x^{(t)} + \mathbf{W}_f \cdot h^{(t-1)})) \\
 i^{(t)} &= \sigma(g(\mathbf{U}_i \cdot x^{(t)} + \mathbf{W}_i \cdot h^{(t-1)})) \\
 \hat{C}^{(t)} &= \tanh(g(\mathbf{U}_C \cdot x^{(t)} + \mathbf{W}_C \cdot h^{(t-1)})) \\
 C^{(t)} &= f^{(t)} \cdot C^{(t-1)} + i^{(t)} \cdot \hat{C}^{(t)} \\
 o^{(t)} &= \sigma(g(\mathbf{U}_o \cdot x^{(t)} + \mathbf{W}_o \cdot h^{(t-1)})) \\
 h^{(t)} &= o^{(t)} \cdot \tanh(C^{(t)})
 \end{aligned}$$

LSTMs keep a cell state $C^{(t)}$ which is transformed with the forget $f^{(t)}$ and input $i^{(t)}$ gates, whose output is in the $[0,1]$ range thanks to a sigmoid non-linearity. The forget gate $f^{(t)}$ determines how much of the previous state $C^{(t-1)}$ we keep, while the input gate $i^{(t)}$ determines how much of the input will be added to the new state $C^{(t)}$. Then, the output gate $o^{(t)}$ is applied to the new cell state to compute $h^{(t)}$. Intuitively, LSTMs decide whether the current time-step information matters or not, while the forget gate decides what can be discarded from the previous cell state.

Gated Recurrent Unit (GRU) [37] often exhibits similar performance as the LSTM but has fewer parameters, which is achieved by combining the input and forget gates into a single update gate $z^{(t)}$, and merging the hidden state and the cell state:

$$\begin{aligned}
 z^{(t)} &= \sigma(g(\mathbf{U}_z \cdot x^{(t)} + \mathbf{W}_z \cdot h^{(t-1)})) \\
 r^{(t)} &= \sigma(g(\mathbf{U}_r \cdot x^{(t)} + \mathbf{W}_r \cdot h^{(t-1)})) \\
 \hat{h}^{(t)} &= \tanh(g(\mathbf{U}_h \cdot x^{(t)} + \mathbf{W}_h \cdot h^{(t-1)} \cdot r^{(t)})) \\
 h^{(t)} &= (1 - z^{(t)}) \cdot h^{(t-1)} + z^{(t)} \cdot \hat{h}^{(t)}
 \end{aligned}$$

As it can be seen in their respective equations, LSTM and GRU compute their outputs $h^{(t)}$ mostly using additive operations between their gates and previous states, which

contrasts with the multiplicative computation in vanilla RNNs (see Equation 9). Such property allows LSTM and GRU to alleviate the problem with vanishing and exploding gradients in RNNs. Figure 0.10(a) depicts the computation flow of vanilla RNN, LSTM and GRU.

RNNs have been successfully applied to many natural language processing and computer vision tasks, such as image captioning [201], video classification [51] or machine translation [190]. Although LSTMs and GRUs were designed to work with long sequences, they are still known to struggle with them, since a single embedding $h^{(t)}$ stores the whole history of previous outputs. RNNs are also slow to train due to their auto-regressive nature both in training and inference (i.e. the output at time-step t can only be computed after all previous time-steps $t = \{1, \dots, t-1\}$ have been obtained). In this direction, recent works have proposed to replace RNNs with models using attention [196] or convolutions [62] to capture temporal relationships, which speed-up training and achieve comparable results. For the purposes of this thesis, we focus on the former, which will be used as a major component for models developed in Chapters 5 and 9.

Attention mechanisms allow neural networks to focus on portions of the input in order to predict an output. Given a set of inputs x of size n (which can represent n local features extracted from an image), and the hidden state $h^{(t)}$ of an RNN at time-step t , the attention mechanism assigns a weight to each input vector x_i as follows:

$$\alpha_i^{(t)} = \frac{e^{f_\phi(h^{(t)}, x_i)}}{\sum_{j=1}^n e^{f_\phi(h^{(t)}, x_j)}} \quad (10)$$

The function f_ϕ computes a similarity score between pairs of vectors (in the example above, the similarity is computed between $h^{(t)}$ and each local feature x_i). This similarity function can take multiple forms, although the most common are additive attention $f_\phi(h^{(t)}, x_i) = v_a^T \cdot \tanh(W_a \cdot [h^{(t)}; x_i])$ [201, 215, 128] (where v_a and W_a are learnable parameters), and dot-product attention $f_\phi(h^{(t)}, x_i) = \frac{h^{(t)T} \cdot x_i}{\sqrt{n}}$ [196]. Finally, the output vector $c^{(t)}$ is obtained by computing a weighted sum of input vectors using the attention coefficients α :

$$c^{(t)} = \sum_{i=1}^n \alpha_i^{(t)} \cdot x_i \quad (11)$$

While the same input representation x_i is often used both to compute and apply attention weights, it is possible to have separate representations of the same input data for each operation. To accommodate for this, the nomenclature for attention mechanisms refers to vectors x_i used in Equation 10 as *keys*, and vectors x_i in Equation 11 are called *values*. The condition vector $h^{(t)}$ used in Equation 10 is known as the *query*. Thus, attention mechanisms operate on triplets of queries, keys and values: first, similarity scores are computed between the *query* and the *keys*, which are then used to obtain $c^{(t)}$ as weighted sum of the *values*. Attention mechanisms have been widely used in the deep learning literature, particularly for machine translation (where attention is computed over words in the input sentence in order to decode each word in the output sentence) or image captioning (where certain image regions are picked in order to predict each word in the output description).

The term of self-attention refers to an attention mechanism that relates different positions of the same data source (e.g., a sequence) to obtain a representation of its own elements. Self-attention can be used as a replacement to recurrent neural networks, where instead of keeping track of a hidden state along a sequence, each element to be predicted in the sequence is represented as a weighted combination of its preceding ones. Self-attention is used in a popular attention-based auto-regressive neural network called the Transformer [196]. Given a source sentence $x = (x_1, \dots, x_n)$, the Transformer decoder conditions the prediction of each word y_t in the target sentence $y = (y_1, \dots, y_m)$ by applying attention over 1) previously predicted words $\hat{y}_{<t}$, and 2) context vectors $e = \{e_1, \dots, e_k\}$ extracted from words in x . First, self-attention is computed by taking \hat{y}_{t-1} as query vector, and $\hat{y}_{<t-1}$ as keys and values. After applying self-attention, attention over words in the input sentence is computed by using the output of self-attention as query and c_t as keys and values. This process is repeated B times, where B is a hyperparameter and denotes the number of transformer layers of the architecture. Figure 0.10(b) illustrates the Transformer decoder architecture.

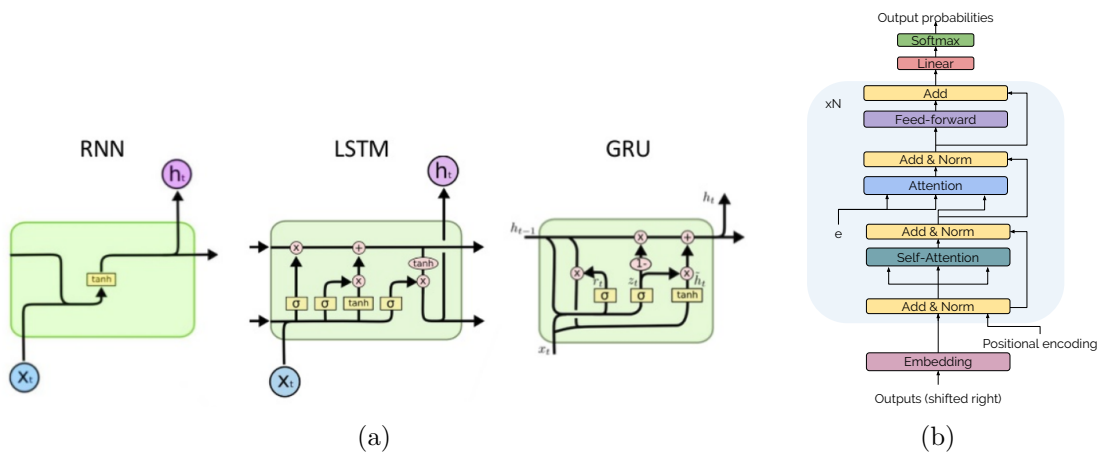


Figure 0.10: (a): Recurrent Architectures: RNN, LSTM and GRU. Figure from [152]. (b): The Transformer Decoder [196].

To achieve parallelization during training, the Transformer is trained with teacher forcing, i.e. during training $\hat{y} = (\text{start}, y_1, \dots, y_{m-1})$, which removes the dependency to previous outputs and allows faster training times. Further, in order for the attention mechanism to account for the word order in the sequence, each word token is augmented with an indicator of the position it occupies (known as *positional encoding*). The Transformer will be used in Chapters 5 and 9 as one of the model architectures to predict image labels, ingredients and cooking instructions from still images.

Part I

Visual Instance Search

Introduction

1

With the proliferation of ubiquitous cameras and Internet technology advances, billions of people are nowadays immersed in the Web sharing and browsing photos and videos. Visual media is currently the most common type of content in social media channels. As of 2019, 300 million photos are uploaded daily on Facebook, and 300 hours of video are uploaded every minute on YouTube¹. Large scale data collections such as these ones are nontrivial to organize in order to efficiently provide relevant results for a user query. For this reason, and even though image search has been broadly explored since the early 90s [175], it has become an important research area in the multimedia and computer vision communities in the past decade. This part of the thesis is dedicated to the problem of content-based image retrieval (CBIR), and proposes strategies that rely in representations extracted from deep neural networks. This chapter introduces the aforementioned task, presents datasets and evaluation metrics and reviews the related work.

Content-based image retrieval

Traditional image search engines typically index visual data based on its surrounding metadata (e.g., titles, descriptions or tags). Metadata is, however, usually provided by the user who uploaded the visual content, thus it may not always be available. Further, textual information may be inconsistent with the visual content, or might not be completely describing its semantics. Such limitations of text-based image retrieval led to the emergence of content-based image retrieval systems, which have greatly advanced in the recent years.

Content-based image retrieval (CBIR) [185] aims at organizing and structuring image datasets based on their content rather than their associated metadata. The most common query technique in CBIR is the Query by example [237], in which the user provides an example image or an image region to base the search upon. Methods presented in this part of the thesis are developed assuming the query is always an image, an image region (i.e. delimited with a bounding box), or a group of images representing the concept of interest. Since the visual query is (in most cases) unknown beforehand, CBIR systems must index database images in such a way that enables their retrieval given any visual query. The representation of content is typically extracted from the image using computer vision techniques, which condense the relevant information in the image into a fix-sized representations that facilitate their indexing.

Early CBIR approaches described image contents using global features based on color [191], shape [153] or texture [44]. Later on, methods relying on the aggregation of local features [127] extracted from regions of interest in the image were proven to be more

¹<https://dustn.tv/social-media-statistics/> (Accessed on February 2019)

effective for CBIR. Recently, since the emergence of deep learning in computer vision, many works in the literature [11, 12, 192, 93, 178, 213] have replaced hand-crafted descriptors with solutions based on off-the-shelf features extracted from a ConvNet, improving performance in retrieval benchmarks. The typical image retrieval engine is composed of two states: (a) an initial highly-scalable ranking mechanism on the full image database and (b) a more computational demanding yet higher-precision reranking mechanism applied to the top retrieved items obtained in the first stage. This reranking mechanism often takes the form of geometric verification and spatial analysis [88, 229, 134, 228], after which the best matching results can be used for query expansion (pseudo-relevance feedback) [7, 39].

Instance Search

Given a CBIR system, the user formulates a visual query for which the system must return a list of images, ranked according to their similarity to the query. Now, what does *similarity* mean? The notion of similarity is application dependent, and should align with the user’s intention. Figure 1.1 shows examples of possible outcomes given the same visual query. While system A seems to be focusing in the main object category (“dog”) that is present in the query image, system B returns images of different breeds of herding dogs, and system C is returning images including *this* specific dog (the fictional character, *Lassie*). None of the three ranked lists is better than the other, in the sense that all results are *relevant* to the query, at different semantic degrees. However, most works on CBIR focus on the latter case, which illustrates the problem of instance-level image retrieval (or instance search), which specifically constraints the search to the particular instance of the query (e.g., a particular object, person or location).

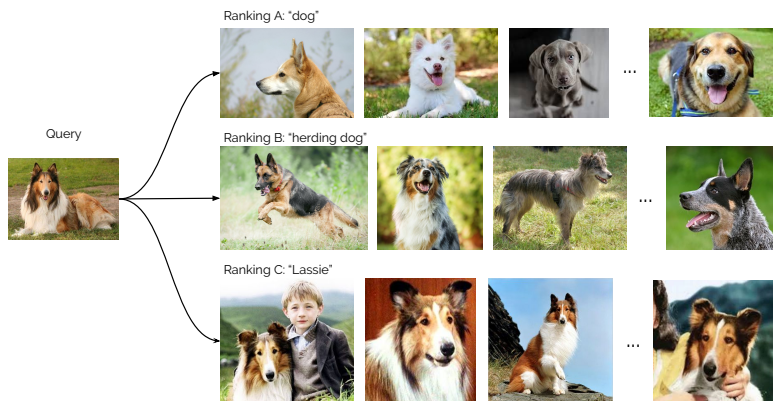


Figure 1.1: Query similarity. Retrieved items for the same visual query using three different search engines.

This part of the thesis presents two instance search methods which use deep neural networks as the source to extract representations describing image contents. Chapter 2 proposes an efficient and scalable solution based on Bag of Words from activations of a ConvNet pre-trained for image classification, and Chapter 3 explores the benefits of using features from a ConvNet trained for object detection, which is fine-tuned for each specific query.

Related Work

This section reviews previous works on instance search, with a focus on aggregation strategies based on hand-crafted features (Section 1.3.1), deep representations (Section 1.3.2) and reranking and query expansion strategies (Section 1.3.3).

Hand-crafted representations

Before the emergence of ConvNets in computer vision, state-of-the-art instance search systems relied on representations based on the aggregation of local hand-crafted features. The most common algorithm for local hand-crafted feature extraction is the Scale-Invariant Feature Transform (SIFT [127]), which was designed to be invariant to image translation, scaling and rotation. This representation encodes the orientation of the image gradient at each keypoint location, which aggregates gradient orientations at different grid locations surrounding the keypoint. Most retrieval pipelines rely on the Hessian Affine keypoint detector [138] to provide relevant locations, and extract features at each of location with SIFT [127] or its improved variant RootSIFT [7]. Then, representations extracted from all keypoints are subsequently aggregated to obtain a fix-length representation. The most widely used aggregation method is the Bag of Words (BoW) [183], which has been subsequently enhanced with sophisticated techniques such as query foreground/background weighting [227], asymmetric distances [234], or larger vocabularies [158, 235]. While BoW only encodes the occurrences of each centroid in the image, other aggregation techniques such as VLAD [89] and Fisher Vectors [157] encode distances between centroid and assigned point coordinates in the quantization space. These result in more informative but dense high-dimensional descriptors, which must be combined with compression methods [235] for fast retrieval. In Chapter 2, we revisit the Bag of Words encoding strategy and demonstrate its suitability to aggregate CNN features instead of hand-crafted ones.

Deep Representations

Several authors have adopted deep representations for image retrieval. The first attempts focused on replacing traditional aggregated hand-crafted local descriptors with features from fully connected layers of a ConvNet pre-trained for image classification. In this context, Babenko et al. [12] showed how such features can reach similar performance to hand-crafted features encoded with Fisher Vectors for image retrieval. Razavian et al. [164] later outperformed the state-of-the-art of ConvNet representations for retrieval using several image sub-patches as input to a pretrained ConvNet to extract features from fully connected layers at different locations of the image. Similarly, Liu et al. [124] used features from fully connected layers evaluated on image sub patches to encode images using Bag of Words.

A second generation of works reported significant gains in performance when switching from fully connected to convolutional layers. Razavian et al. [178] performed spatial max pooling on the feature maps of a convolutional layer of a pre-trained ConvNet to produce a descriptor of the same dimension as the number of filters of the layer. Babenko and Lempitsky [11] proposed a compact descriptor based on sum pooling of convolutional feature maps preprocessed with a Gaussian center prior. Tolias et al. [192] introduce a feature representation based on the integral image to quickly max pool features from local

patches of the image and encode them in a compact representation. The recent work by Kalantidis et al. [93] proposed non-parametric spatial and channel-wise weighting schemes, which were applied directly to the convolutional features before sum pooling.

Several authors have tried to exploit local information in images by passing multiple image sub patches through a ConvNet to obtain local features from either fully connected [164, 124] or convolutional [68] layers, which are in turn aggregated using techniques like average pooling [164], BoW [124], or VLAD [68]. Although many of these methods perform well in retrieval benchmarks, they are significantly more computationally costly since they require an independent ConvNet forward pass for each of the considered image patches, which slows down indexing and feature extraction at retrieval time. An alternative approach is to extract convolutional features for the full image and treat the activations of the different neuron arrays across all feature maps as local features. This way, a single forward pass of the entire image through the ConvNet is enough to obtain the activations of its local patches. Following this approach, Ng et al. [149] propose to use VLAD [89] encoding of features from convolutional layers to produce a single image descriptor. Arandjelovic et al. [6] choose to adapt a ConvNet with a layer especially trained to learn the VLAD parameters.

The contributions presented in this part of the thesis also use convolutional features extracted from a pre-trained ConvNet. In Chapter 2, we propose using a sparse, high-dimensional encoding that better represents local image features, particularly in challenging instance search scenarios where the target object is not the primary focus of the image. Chapter 3 proposes the use of an object detection ConvNet, to extract both global and local convolutional features in a single forward pass.

Reranking Strategies

State of the art retrieval solutions often rely on an initial search over the entire database, combined with a posterior reranking stage over the top-K elements retrieved in the first step. The reranking stage usually involves methodologies based on spatial verification. Zhou et al. [233] propose a fast spatial verification technique which benefits from the BoW encoding to choose tentative matching feature pairs between the query and the target image. Zhang et al. [228] introduce an elastic spatial verification step based on triangulated graph model. Nguyen et al. [150] propose a solution based on deformable parts models (DPM) [58] to rerank a BoW-based baseline. They train a neural network on several query features to learn the weights to fuse the DPM and BoW scores.

Methods relying on deep representations have also explored spatial verification stages. Razavian et al. [178] achieve a remarkable increase in performance by applying a spatial search strategy over an arbitrary grid of windows at different scales. Although they report high performance in several retrieval benchmarks, their proposed approach is very computationally costly and does not scale well to larger datasets and real-time search scenarios. Tolias et al. [192] introduce a local analysis of multiple image patches, which is only applied to the top elements of an initial ranking. They propose an efficient workaround for sub patch feature pooling based on integral images, which allows them to quickly evaluate many image windows. Their approach improves their baseline ranking and also provides approximate object localizations. They apply query expansion using images from the top of the ranking after the reranking stage, although they do not use the obtained object locations in any way to improve retrieval performance.

In this direction, in Chapter 2 we take advantage of the BoW vocabulary to obtain representations for image patches, which allow fast spatial search for reranking. For the same purpose, in Chapter 3 we use representations from object proposals obtained with an object detector.

Datasets

We use the following datasets to evaluate the performance of our algorithms:

Oxford Buildings [158] contains 5,063 still images, including 55 query images of 11 different buildings in Oxford. A bounding box surrounding the target object is provided for query images. An additional 100,000 distractor images are also available for the dataset. We refer to the original and extended versions of the dataset as Oxford 5k and Oxford 105k, respectively.

Paris Buildings [159] contains 6,412 still images collected from Flickr including query images of 12 different Paris landmarks with associated bounding box annotations. A set of 100,000 images is added to the original dataset (Paris 6k) to form its extended version (Paris 106k).

TRECVID Instance Search 2013 [184] contains 244 video files (464 hours in total), each containing a week’s worth of BBC EastEnders programs. Each video is divided in different shots of short duration (between 5 seconds and 2 minutes). We perform uniform keyframe extraction at 1/4 fps. The dataset also includes 30 queries and provides 4 still images for each of them (including a binary mask of the object location). In our experiments, we use a subset of this dataset that contains only those keyframes that are positively annotated for at least one of the queries. The dataset, which we will refer to as the TRECVID INS subset, is composed of 23,614 keyframes.

Figure 1.2 includes three examples of query objects from each of the three datasets. While the three datasets are curated for the task of instance search, query objects from both Oxford and Paris Buildings are roughly located in the center of the image, and are much larger in proportion to the image size than those in TRECVID. Further, the context surrounding an object to be found can significantly vary across videos in TRECVID (a *wooden bench* or a *no-smoking sign* can be found in many scenarios), while it rarely does in Oxford and Paris (buildings do not move). The particularities of the different datasets must be taken into account when designing retrieval methods that are suitable for them. As we will see in the remainder of this chapter, high-performing approaches for Oxford and Paris can fail when directly applied to TRECVID due to the nature of the datasets.

Metrics

We evaluate the obtained rankings with the mean Average Precision (mAP) metric, commonly used in information retrieval:

$$mAP = \frac{\sum_{q=1}^Q AP(q)}{Q} \quad (1.1)$$



Figure 1.2: Retrieval datasets. Query examples from Paris buildings (top, 1-3), Oxford buildings (top, 4-6) and TRECVID INS 2013 (bottom).

where $AP(q)$ is the Average Precision for each query q , which is calculated as:

$$AP(q) = \frac{\sum_{k=1}^K P(k) \times rel(k)}{N} \quad (1.2)$$

Bags of Deep Visual Words

2

Before the consolidation of ConvNets as powerful feature extractors for computer vision tasks, most of the best performing instance search systems [150, 228, 233, 235] were based on aggregating local hand-crafted features (e.g., SIFT) using bag of words encoding [183] to produce very high-dimensional sparse image representations. Such high-dimensional sparse representations have several benefits over their dense counterparts. High-dimensionality means they are more likely to be linearly separable, while having relatively few non-zero elements makes them efficient both in terms of storage (only non-zero elements need to be stored), and computation (only non-zero elements need to be visited). Sparse representations can handle varying information content, and are less likely to interfere with one another when pooled. From an information retrieval perspective, sparse representations can be stored in inverted indices, which facilitates efficient selection of images that share features with a query. Furthermore, there is considerable evidence that biological systems make extensive use of sparse representations for sensory information [107, 197]. Empirically, sparse representations have repeatedly demonstrated to be effective in a wide-range of vision and machine learning tasks.

In this chapter, inspired by advances in ConvNet-based descriptors for image retrieval, yet still focusing on instance search, we revisit the Bag of Words encoding scheme using local features from convolutional layers of a ConvNet. This work presents three contributions:

- We propose a sparse visual descriptor based on a Bag of Local Convolutional Features (BLCF), which allows fast image retrieval by means of an inverted index.
- We introduce the *assignment map* as a new compact representation of the image, which maps pixels in the image to their corresponding visual words. The assignment map allows fast composition of a BoW descriptor for any region of the image.
- We take advantage of the scalability properties of the assignment map to perform a local analysis of multiple regions of the image for reranking, followed by a query expansion stage using the obtained object localizations.

Using this approach, we present an image retrieval system that achieves competitive performance in CBIR benchmarks and outperforms current state-of-the-art ConvNet descriptors at the task of instance search.

The remainder of this chapter is structured as follows. Section 2.1 introduces the proposed framework for BoW encoding of deep local features. Section 2.2 explains the details of our retrieval system, including the local reranking and query expansion stages. Section 2.3 presents experimental results on three different retrieval benchmarks (Oxford Buildings,

Paris Buildings, and a subset of TRECVID INS 2013), as well as a comparison to five other state-of-the-art approaches. Section 2.4 presents the conclusion of this work.

Bags of Deep Local Features

The proposed pipeline for feature extraction uses the activations at different locations of a convolutional layer in a pre-trained ConvNet as local features. A ConvNet trained for a classification task is typically composed of a series of convolutional layers, followed by a series of fully connected layers, connected to a softmax layer that produces the inferred class probabilities. To obtain a fixed-sized output, the input image to a ConvNet is usually resized to be square. However, several authors using ConvNets for retrieval [192, 93] have reported performance gains by retaining the aspect ratio of the original images. We therefore discard the softmax and fully connected layers of the architecture and extract ConvNet features maintaining the original image aspect ratio.

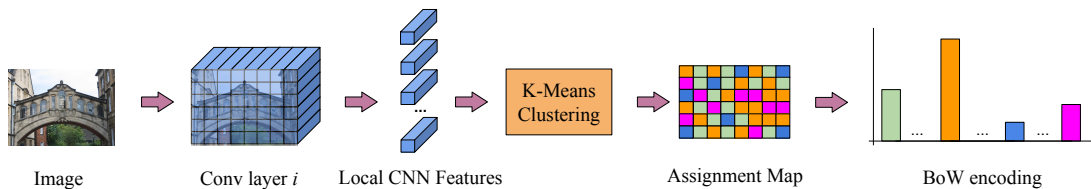


Figure 2.1: The Bag of Local Convolutional Features pipeline (BLCF).

Each convolutional layer in the network has D different $N \times M$ feature maps, which can be viewed as $N \times M$ descriptors of dimension D . Each of these descriptors contains the activations of all neurons in the convolutional layer sharing the same receptive field. This way, these D -dimensional features can be seen as local descriptors computed over the region corresponding to the receptive field of an array of neurons. With this interpretation, we can treat the ConvNet as a dense local feature extractor and use any existing aggregation technique to build a single image representation.

We propose to use the Bag of Words model to encode the local convolutional features of an image into a single vector. Although more elaborate aggregation strategies have been shown to outperform BoW-based approaches for some tasks in the literature [89, 157], Bag of Words encodings produce sparse high-dimensional codes that can be stored in inverted indices, which are beneficial for fast retrieval. Moreover, BoW-based representations are faster to compute, easier to interpret, more compact, and provide all the benefits of sparse high-dimensional representations.

Bag of Words models require constructing a visual codebook to map vectors to their nearest centroid. We use k -means on local ConvNet features to fit this codebook. Each local feature in the convolutional layer is then assigned its closest visual word in the learned codebook. This procedure generates the *assignment map*, i.e. a 2D array of size $N \times M$ that relates each local convolutional feature with a visual word. The assignment map is therefore a compact representation of the image which relates each pixel of the original image with its visual word with a precision of $(\frac{W}{N}, \frac{H}{M})$ pixels, where W and H are the width and height of the original image. This property allows us to quickly generate the BoW vectors of not only the full image, but also its local patches. We describe the use of this property in our work in Section 2.2.

Figure 2.1 shows the pipeline of the proposed approach. The described method encodes the image into a sparse high dimensional descriptor, which will be used as the image representation for retrieval.

Image Retrieval Pipeline

This section describes the image retrieval pipeline, which consists of an initial ranking stage, followed by a local reranking, and query expansion.

(a) Initial search: The initial ranking is obtained using the cosine similarity between the BoW vector of the query image and the BoW vectors of the images in the database. We use a sparse matrix based inverted index and GPU-based sparse matrix multiplications to allow fast retrieval. The image list is then sorted based on the cosine similarity of its elements to the query. We use two types of image search based on the query information that is used:

- **Global search (GS):** The BoW vector of the query is built with the visual words of all the local features in the convolutional layer extracted for the query image.
- **Local search (LS):** The BoW vector of the query contains only the visual words of the features that fall inside the bounding box that is provided for each of the query images.

(b) Local reranking (R): After the initial search, the top K images in the ranking are locally analyzed and reranked based on a localization score. The local reranking is based on computing a similarity score between the query bounding box and a set of image locations S extracted from each the K images. We construct S with sliding windows with 50% of overlap of sizes comprising all possible combinations of width $w \in \{W, \frac{W}{2}, \frac{W}{4}\}$ and height $h \in \{H, \frac{H}{2}, \frac{H}{4}\}$, where W and H are the width and height of the assignment map. We use a sliding window strategy directly on the assignment map to build the BoW vector for each image region in S .

We perform a first filtering strategy to discard those windows whose aspect ratio is too different to the aspect ratio of the query. Let the aspect ratio of the query bounding box be $AR_q = \frac{W_q}{H_q}$ and $AR_w = \frac{W_w}{H_w}$ be the aspect ratio of the window. The score for window w is defined as $score_w = \frac{\min(AR_w, AR_q)}{\max(AR_w, AR_q)}$. All windows with a score lower than a threshold th are discarded.

For each of the remaining windows, we construct the BoW vector representation and compare it with the query representation using cosine similarity. The window with the highest cosine similarity is assigned as the new similarity score for the image (max pooling of scores).

We also enhance the BoW window representation with spatial pyramid matching [104] with $L = 2$ resolution levels (i.e. the full window and its 4 sub regions). We construct the BoW representation of all sub regions at the 2 levels, and weight their contribution to the similarity score with inverse proportion to the resolution level of the region. The cosine similarity of a sub region r to the corresponding query sub region is therefore weighted

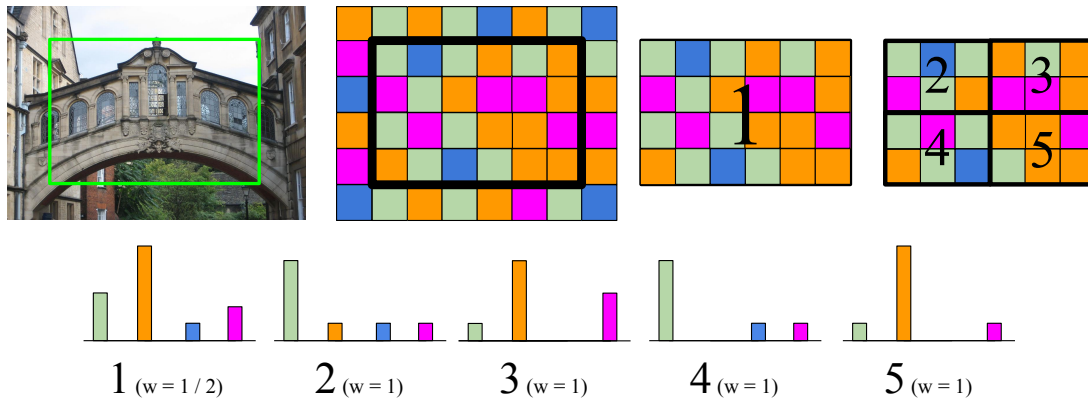


Figure 2.2: Spatial Pyramid representation of an object region, highlighted with a green bounding box.

by $w_r = \frac{1}{2^{(L-l_r)}}$, where l_r is the resolution level of the region r . Figure 2.2 depicts the described approach.

With this procedure, the top K elements of the initial ranking are sorted based on the cosine similarity of their regions with respect to the query region, and also provides the region with the highest score as a rough localization of the object.

(c) Query expansion. We investigate two query expansion strategies based on global and local BoW descriptors, where the top N images of the ranking are used to augment the query:

- Global query expansion (GQE): The BoW vectors of the N images at the top of the ranking are averaged together with the BoW of the query to form an updated representation for the query. GQE can be applied either before or after the local reranking stage.
- Local query expansion (LQE): Locations obtained in the local reranking step are used to mask out the background and build the BoW descriptor of only the region of interest of the N images at the top of the ranking. These BoW vectors are averaged together with the BoW of the query bounding box.

In both cases, the updated BoW vector is used to perform a second search.

Experiments

In this section, we describe the implementation details of our method, including an analysis of each of its components and a comparison with state-of-the-art image retrieval methods.

Preliminary experiments

Feature extraction was performed using *Caffe* [91] and the VGG16 network [182] pre-trained with ImageNet [177]. We extracted features from the last three convolutional

	<i>conv5_1</i>	<i>conv5_2</i>	<i>conv5_3</i>
$N \times M$ raw	0.641	0.626	0.498
$2N \times 2M$ interpolated	0.653	0.638	0.536
$2N \times 2M$ raw	0.620	0.660	0.540

Table 2.1: Layer Comparison. Mean average precision (mAP) on Oxford 5k using different convolutional layers of VGG16, comparing the performance of different feature map resolutions (both raw and interpolated). The size of the codebook is 25,000 in all experiments.

layers (*conv5_1*, *conv5_2* and *conv5_3*) and compared their performance on the Oxford 5k dataset. We experimented on different image input sizes: 1/3 and 2/3 of the original image. Following several other authors [11, 93], we l_2 -normalize all local features, followed by PCA, whitening, and a second round of l_2 -normalization. The PCA models were fit on the same dataset as the test data in all cases.

Unless stated otherwise, all experiments used a visual codebook of 25,000 centroids fit using the (L^2 -PCA- L^2 transformed) local convolutional features of all images in the same dataset (1.7M and 2.15M for Oxford 5k and Paris 6k, respectively). We tested three different codebook sizes (25,000; 50,000 and 100,000) on the Oxford 5k dataset, and chose the 25,000 centroids one because of its higher performance.

Table 2.1 shows the mean average precision on Oxford 5k for the three different layers and image sizes. The combination of the layers by concatenation did not provide any gain. We also consider the effect of applying bilinear interpolation of the feature maps prior to the BoW construction, as a fast alternative to using a larger input to the ConvNet. Our experiments show that all layers benefit from feature map interpolation. Our best result was achieved using *conv5_2* with full size images as input. However, we discarded this configuration due to its memory requirements: on a Nvidia GeForce GTX 970, we found that feature extraction on images rescaled with a factor of 1/3 was 25 times faster than on images twice that size. For this reason, we resize all images to 1/3 of their original size and use *conv5_1* interpolated feature maps.

Inspired by the boost in performance of the Gaussian center prior in SPoC features [11], we also apply a weighting scheme on the visual words of an image to provide more importance to those belonging to the central part of the image. All results in Table 2.1 are obtained using features weighted with a normalized inverse distance map to the image center which, for *conv5_1* in Oxford 5k, increases mAP from 0.626 to 0.653.

Query augmentation

Previous works [7, 194] demonstrated how simple data augmentation strategies can improve the performance of an instance search system. Some of these apply augmentation strategies at the database side, which can be prohibitively costly for large datasets. For this reason, we use data augmentation on the query side only. We explore two different strategies to enrich the query before visual search: a horizontal flip (or mirroring) and a zoomed central crop (ZCC) on an image enlarged by 50%. The feature vectors they produce are added together to form a single BoW descriptor. Table 2.2 shows the impact of incrementally augmenting the query with each one of these transformations.

	Query	+ Flip	+ ZCC	+ Flip + ZCC
GS	0.653	0.662	0.695	0.697
LS	0.738	0.746	0.758	0.758

Table 2.2: Query Augmentation Results. mAP on Oxford 5k for the two different types of query augmentation: the flip and the zoomed central crop (ZCC). $2\times$ interpolated *conv5_1* features are used in all cases.

	Oxford 5k		Paris 6k	
		+ Q_{aug}		+ Q_{aug}
GS	0.653	0.697	0.699	0.754
LS	0.738	0.758	0.820	0.832
GS + R	0.701	0.713	0.719	0.752
LS + R	0.734	0.760	0.815	0.828
GS + GQE	0.702	0.730	0.774	0.792
LS + GQE	0.773	0.780	0.814	0.832
GS + R + GQE	0.771	0.772	0.801	0.798
LS + R + GQE	0.769	0.793	0.807	0.828
GS + R + LQE	0.782	0.757	0.835	0.795
LS + R + LQE	0.788	0.786	0.848	0.833

Table 2.3: mAP on Oxford 5k and Paris 5k for the different stages in the pipeline introduced in Section 2.2. The Q_{aug} additional columns indicate the results when the query is augmented with the transformations introduced in Section 2.3.2.

We find that all the studied types of query augmentation consistently improve the results, for both global and local search. ZCC provides a higher gain in performance compared to flipping alone. ZCC generates an image of the same resolution as the original, which contains the center crop at a higher resolution. Objects from the Oxford dataset tend to be centered, which explains the performance gain when applying ZCC.

Reranking and query expansion

We apply the local reranking (R) stage on the top $K = 100$ images in the initial ranking, using the sliding window approach described in Section 2.2. The presented aspect ratio filtering is applied with a threshold $th = 0.4$, which was chosen based on a visual inspection of results on a subset of Oxford 5k. Query expansion is later applied considering the top-10 images of the resulting ranking. This section evaluates the impact in performance of both reranking and query expansion stages. Table 2.3 contains the results for the different stages in the pipeline for both simple and augmented queries (referred to as Q_{aug} in the table).

The results indicate that the local reranking is only beneficial when applied to a ranking obtained from a search using the global BoW descriptor of the query image (GS). This is consistent with the work by Tolias et al. [192], who also apply a spatial reranking



Figure 2.3: BLCF Ranking examples. Examples of the top-ranked images and localizations based on local CNN features encoded with BoW. Top row: *The Christ Church* from the Oxford Buildings dataset; middle row: *The Sacre Coeur* from Paris Buildings; bottom row: query 9098 (*a parking sign*) from TRECVID INS 2013.

followed by query expansion to a ranking obtained with a search using descriptors of full images. They achieve a mAP of 0.66 in Oxford 5k, which is increased to 0.77 after spatial reranking and query expansion, while we reach similar results (e.g., from 0.652 to 0.769). However, our results indicate that a ranking originating from a local search (LS) does not benefit from local reranking. Since the BoW representation allows us to effectively perform a local search (LS) in a database of full indexed images, we find the local reranking stage applied to LS to be redundant in terms of the achieved quality of the ranking. However, the local reranking stage does provide with a rough localization of the object in the images of the ranking, as depicted in Figure 2.3. We use this information to perform query expansion based on local features (LQE).

Results indicate that query expansion stages greatly improve performance in Oxford 5k. We do not observe significant gains after reranking and QE in the Paris 6k dataset, although we achieve our best result with LS + R + LQE.

In the case of augmented queries ($+Q_{aug}$), we find query expansion to be less helpful in all cases, which suggests that the information gained with query augmentation and the one obtained by means of query expansion strategies are not complementary.

Figure 2.3 shows examples of some of the rankings produced by our system on the three different datasets.

Comparison with the state-of-the-art

We compare our approach with other works using ConvNet-based representations on the Oxford and Paris datasets. Table 2.4 includes the best result for each approach in the literature. Our performance using global search (GS) is comparable to that of

	Oxford		Paris	
	5k	105k	6k	106k
Ng et al. [149]	0.649	-	0.694	-
Razavian et al. [178]	0.844	-	0.853	-
SPoC [11]	0.657	0.642	-	-
R-MAC [192]	0.668	0.616	0.830	0.757
CroW [93]	0.682	0.632	0.796	0.710
uCroW [93]	0.666	0.629	0.767	0.695
GS	0.652	0.510	0.698	0.421
LS	0.739	0.593	0.820	0.648
LS + Q_{aug}	0.758	0.622	0.832	0.673
CroW + GQE [93]	0.722	0.678	0.855	0.797
R-MAC + R + GQE [192]	0.770	0.726	0.877	0.817
LS + GQE	0.773	0.602	0.814	0.632
LS + R + LQE	0.788	0.651	0.848	0.641
LS + R + GQE + Q_{aug}	0.793	0.666	0.828	0.683

Table 2.4: Comparison to state-of-the-art ConvNet representations (mAP). Results in the lower section consider reranking and/or query expansion.

Ng et al. [149], which is the one that most resembles our approach. However, they achieve this result using raw VLAD features, which are more expensive to compute and, being a dense high-dimensional representation, do not scale as well to larger datasets. Similarly, Razavian et al. [178] achieve the highest performance of all approaches in both the Oxford and Paris benchmarks by applying a spatial search at different scales for all images in the database. Such approach is prohibitively costly when dealing with larger datasets, especially for real-time search scenarios. Our BoW-based representation is highly sparse, allowing for fast retrieval in large datasets using inverted indices, and achieves consistently high mAP in all tested datasets.

We find the usage of the query bounding box to be extremely beneficial in our case for both datasets. The authors of SPoC [11] are the only ones who report results using the query bounding box for search, finding a decrease in performance from 0.589 to 0.531 using raw SPoC features (without center prior). This suggests that sum pooled convolutional features [11] are less suitable for instance level search in datasets where images are represented with global descriptors.

We also compare our local reranking and query expansion results with similar approaches in the state-of-the-art. The authors of R-MAC [192] apply a spatial search for reranking, followed by a query expansion stage, while the authors of CroW [93] only apply query expansion after the initial search. Our proposed approach also achieves competitive results in this section, achieving the best result for Oxford 5k.

Experiments on TRECVID INS

In this section, we compare the Bag of Local Convolutional Features (BLCF) with the sum pooled convolutional features proposed in several works in the literature. We use our own implementation of the *uCroW* descriptor from [93] and compare it with BLCF for the TRECVID INS subset. For the sake of comparison, we test our implementation

of sum pooling using both our chosen ConvNet layer and input size (*conv5_1* and $1/3$ image size), and the ones reported in [93] (*pool5* and full image resolution). For the BoW representation, we train a visual codebook of 25,000 centroids using 3M local features chosen randomly from the INS subset. Since the nature of the TRECVID INS dataset significantly differs from that of the other ones used so far (see Figure 1.2), we do not apply center prior to the features in any case, to avoid down weighting local features from image areas where the objects might appear. Table 2.5 compares sum pooling with BoW in Oxford, Paris, and TRECVID subset datasets. As stated in earlier sections, sum pooling and BoW have similar performance in Oxford and Paris datasets. For the TRECVID INS subset, however, Bag of Words significantly outperforms sum pooling, which demonstrates its suitability for challenging instance search datasets, in which queries are not centered and have variable size and appearance. We also observe a different behavior when using the provided query object locations (LS) to search, which was highly beneficial in Oxford and Paris datasets, but does not provide any gain in TRECVID INS. We hypothesize that the fact that the size of the instances is much smaller in TRECVID than in Paris and Oxford datasets causes this drop in performance. Global search (GS) achieves better results on TRECVID INS, which suggests that query instances are in many cases correctly retrieved due to their context. Figure 2.4 shows examples of the queries for which the local search outperforms the global search. Interestingly, we find these particular objects to appear in different contexts in the database. In these cases, the usage of the local information is crucial to find the query instance in unseen environments. For this reason, we compute the distance map of the binary mask of the query, and assign a weight to each position of the assignment map with inverse proportion to its value in the distance map. This way, higher weights are assigned to the visual words of local features near the object.

We find this scheme, referred to as weighted search (WS), to be beneficial for most of the queries, suggesting that, although context is necessary, emphasizing the object information in the BoW descriptor is beneficial.

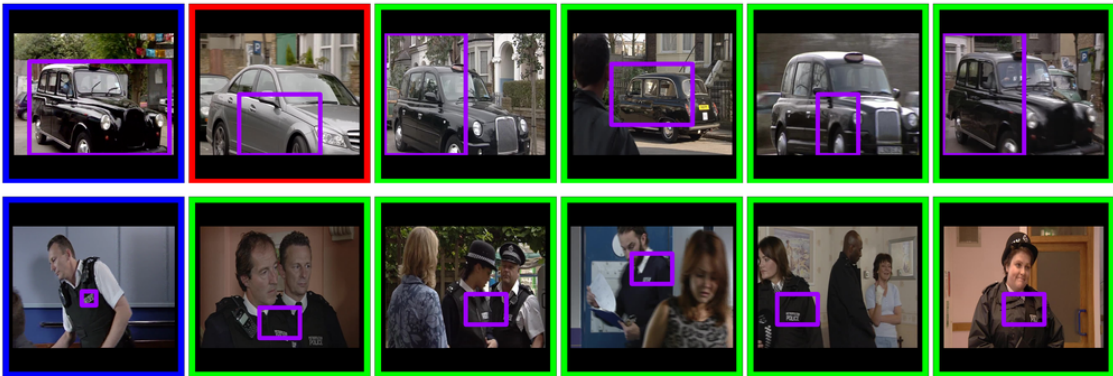


Figure 2.4: TRECVID Ranking Examples. Top 5 rankings for queries 9072 (top) and 9081 (bottom) of the TRECVID INS 2013 dataset.

We finally apply the local reranking and query expansion stages introduced in Section 2.2 to the baseline rankings obtained for the TRECVID INS subset. Since in this dataset we are dealing with objects whose appearance can significantly change in different keyframes, we decided not to filter out windows based on aspect ratio similarity. Additionally, we do not apply the spatial pyramid matching, since some of the query instances are too

small to be divided in sub regions. After reranking, we apply the distance map weighting scheme to the locations obtained for the top 10 images of the ranking and use them to do weighted query expansion (WQE).

Results are consistent with those obtained in the experiments in Oxford and Paris datasets: although the local reranking does not provide significant improvements (WS: 0.350, WS + R: 0.348 mAP), the query expansion stage is beneficial when applied after WS (WS + GQE: 0.391 mAP), and provides significant gains in performance after local reranking (WS + R + GQE: 0.442 mAP) and after local reranking using the obtained localizations (WS + R + WQE: 0.452 mAP).

		Oxford 5k	Paris 6k	INS 23k
BoW	GS	0.650	0.698	0.323
	WS	0.693	0.742	0.350
	LS	0.739	0.819	0.295
Sum pooling (as ours)	GS	0.606	0.712	0.156
	WS	0.638	0.745	0.150
	LS	0.583	0.742	0.097
Sum pooling (as in [93])	GS	0.672	0.774	0.139
	WS	0.707	0.789	0.146
	LS	0.683	0.763	0.120

Table 2.5: Sum pooling vs BoW on Oxford, Paris and TRECVID INS subset.

Conclusion

In this chapter, we proposed an aggregation strategy based on Bag of Words to encode features from convolutional neural networks into a sparse representations for instance search. We demonstrated the suitability of these bags of local convolutional features, achieving competitive performance with respect to other ConvNet-based representations in Oxford and Paris benchmarks, while being more scalable in terms of index size, cost of indexing, and search time. We also compared our BoW encoding scheme with sum pooling of ConvNet features for instance search in the far more complex and challenging TRECVID instance search task, and demonstrated that our method consistently and significantly performs better. This encouraging result suggests that the BoW encoding, as a virtue of being high dimensional and sparse, is more robust to scenarios where only a small number of features in the target images are relevant to the query. Our method does, however, appear to be more sensitive to large numbers of distractor images than methods based on sum and max pooling (SPoC, R-MAC, and CroW). We speculate that this may be because the distractor images are drawn from a different distribution to the original dataset, and may therefore require a larger codebook to better represent the diversity in the visual words.

Object Detectors for Instance Search

3

In both the previous chapter and [178, 192], local image features are extracted for a predefined arbitrary set of bounding boxes of different scales and aspect ratios. Deep features for these regions are extracted by cropping the image [178], the convolutional features [192], or the previously constructed *assignment map*, proposed in Chapter 2. These features can be aggregated together to compose a global image representation [192] or can be used for spatial search, where each bounding box representation is compared to that of the query instance in order to find the optimal location in the image that contains the query (Chapter 2). In this chapter, we investigate whether better representations for image regions can be obtained from ConvNets trained for the task of object detection instead of classification. Modern object detection architectures [167] are trained in an end-to-end manner to simultaneously learn prominent object locations and category scores. Once trained, the output of the model is a list of object locations ranked by both objectness and category scores. Simply using a threshold on the scores of elements in the ranklist results in a small set of regions that are likely to contain objects of interest in the image. The set of highly scored regions typically covers the relevant objects in the image, thus it can significantly change from one image to the other. In this chapter, we argue that these object locations can replace the arbitrary set of bounding boxes used both in [178, 192] and the previous chapter in order to extract region descriptors. Further, we hypothesize that features learned in a network trained for object detection are suitable to the instance search task, since they have been specifically optimized to recognize object instances. Further, we quantify the benefits of fine tuning the object detection network to build a specialized instance search system trained to recognize a fixed known set of queries.

This work explores the suitability of both off-the-shelf and fine-tuned features from an object detection ConvNet for the task of instance retrieval. We make the following three contributions:

- We propose to use a pre-trained ConvNet for object detection to extract convolutional features both at global and local scale in a single forward pass of the image through the network.
- We explore simple spatial reranking strategies, which take advantage of the locations learned by a Region Proposal Network (RPN) to provide a rough object localization for the top retrieved images of the ranking.
- We analyze the impact of fine-tuning an object detection CNN for the same instances one wants to query in the future. We find such a strategy to be suitable for learning better image representations.

The remainder of this chapter is structured as follows. Section 3.1 gives an overview on the state of the art for object detection with neural networks. Section 3.2 introduces the representations for images and regions given a trained object detector. Section 3.3 describes the details involved in the fine tuning of an object detector for instance search. Section 3.4 explains the retrieval pipeline that we followed, and Section 3.5 includes the performed experiments on three different image retrieval benchmarks as well as the comparison to other state of the art ConvNet-based instance search systems. Finally, Section 3.6 draws the conclusions of this work.

ConvNets for Object Detection

Many works in the literature have proposed ConvNet-based object detection pipelines. This section reviews the most common methods object detection, which we classify in two categories: two-stage and one-stage.

Two-stage detectors first extract a reduced set of prominent object locations (commonly referred to as *object proposals*), which are subsequently classified among the C possible labels in the dataset. Girshick et al. presented R-CNN [64], a version of Krizhevsky’s AlexNet [102], fine-tuned for the Pascal VOC Detection dataset [54]. Instead of full images, the regions of an object proposal algorithm [195] were used as inputs to the network. At test time, fully connected layers for all windows were extracted and used to train a bounding box regressor and classifier. Since then, R-CNN has witnessed great improvements, both in terms of accuracy and speed. He et al. proposed SPP-net [80], which used a Spatial Pyramid based pooling layer to improve classification and detection performance. Additionally, they significantly decreased computational time by pooling region features from convolutional features instead of forward passing each region crop through all layers in the ConvNet. This way, the computation of convolutional features is shared for all regions in an image. Girshick later proposed Fast R-CNN [63], which used the same speed strategy as SPP-net but also replaced the post-hoc training of SVM classifiers and box regressors with an end-to-end training solution. Ren et al. introduced Faster R-CNN [167], which removed the object proposal dependency of former object detection ConvNets by introducing a Region Proposal Network (RPN). In Faster R-CNN, the RPN shares features with the object detection network in [63] to simultaneously learn prominent object proposals and their associated class probabilities.

One-stage detectors do not rely on object proposal extraction. Instead, they directly score a dense grid of overlapping bounding boxes. One-stage architectures such as SSD [121] and YOLO [165] were proposed in the literature, which achieve significantly faster runtimes with respect to their two-stage counterparts, at the expense of a drop in accuracy. Until recently, two-stage detectors have dominated the state of the art in object detection. A novel loss that accounts for class imbalance in one-stage detectors [115] has been shown to surpass the performance of two-stage methods [167], while matching the speed of previous one-stage detectors [121, 165].

In this work, we take advantage of the end-to-end two-stage object detection architecture of Faster R-CNN to extract both image and region features for instance search.

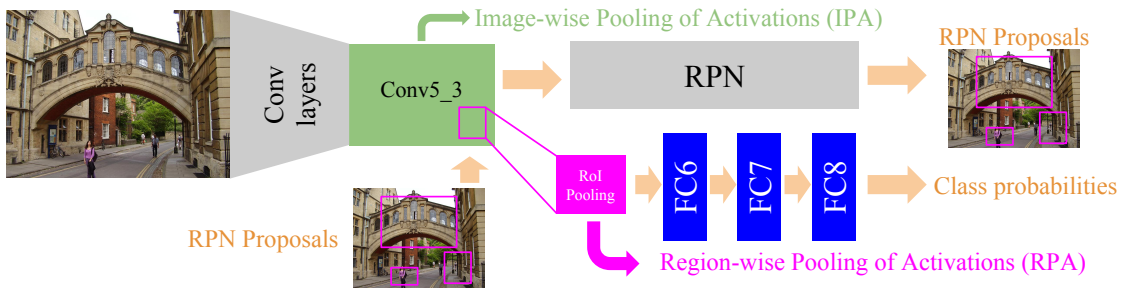


Figure 3.1: Image- and region-wise descriptor pooling Faster R-CNN.

Deep Representations for Images and Regions

This chapter explores the suitability of using features from an object detection ConvNet for the task of instance search. In our setup, query instances are defined by a bounding box over the query images. We choose the architecture and pre-trained models of Faster R-CNN [167] and use it as a feature extractor at both global and local scales. Faster R-CNN is composed of two branches that share convolutional layers. The first branch is a Region Proposal Network that learns a set of window locations, and the second one is a classifier that learns to label each window as one of the classes in the training set.

Similarly to other works [11, 192, 93] our goal is to extract a compact image representation built from the activations of a convolutional layer in a ConvNet. Since Faster R-CNN operates at both global and local scales, we propose the following strategies of feature pooling:

Image-wise pooling of activations (IPA). In order to construct a global image descriptor from Faster R-CNN layer activations, one can choose to ignore all layers in the network that operate with object proposals and extract features from the last convolutional layer. Given the activations of a convolutional layer extracted for an image, we aggregate the activations of each filter response to construct an image descriptor of the same dimension as the number of filters in the convolutional layer. Both max and sum pooling strategies are considered and compared in Section 3.5.2.

Region-wise pooling of activations (RPA). After the last convolutional layer, Faster R-CNN implements a region pooling layer that extracts representations for each of the object proposals learned by the RPN. This way, for each one of the window proposals, it is possible to compose a descriptor by aggregating the activations of that window in the RoI pooling layer, giving raise to the region-wise descriptors. For the region descriptor, both max and sum pooling strategies are tested as well.

Figure 3.1 shows a schematic of the Faster R-CNN architecture and the two types of descriptor pooling described above.

As in the previous chapter, and following several other authors [11, 93], sum-pooled features are l_2 -normalized, followed by whitening and a second round of l_2 -normalization, while max-pooled features are just l_2 -normalized once (no whitening).

Fine-tuning Faster R-CNN

This chapter explores the suitability of fine-tuning Faster R-CNN to 1) obtain better feature representations for image retrieval and 2) improve the performance of spatial analysis and reranking. To achieve this, we choose to fine tune Faster R-CNN to detect the query objects to be retrieved by our system. This way, we modify the architecture of Faster R-CNN to output the regressed bounding box coordinates and the class scores for each one of the query instances of the tested datasets.

In our experiments, we explore two modalities of fine-tuning:

- **Fully-Connected only** (FT_{fc}): Only the weights of the fully connected layers in the classification branch are updated (i.e. the convolutional layers and the RPN are left unchanged).
- **All layers after *conv_2*** (FT_{all}): Weights of all layers after the first two convolutional layers are updated. This way, convolutional features, RPN proposals and fully connected layers are modified and adapted to the query instances.

In the case of Oxford and Paris, we modify the last classification layer in the network to return 12 class probabilities (11 buildings in the dataset, plus an extra class for the background). We also modify the bounding box regression layer to output coordinate offsets for objects belonging to each of the output categories. We use the 5 images provided for each one of the buildings and their bounding box locations as training data. Additionally, we augment the training set by performing a horizontal flip on the training images ($11 \times 5 \times 2 = 110$ training images in total). For INS 13, we have 30 different query instances, with 4 images each, giving raise to $30 \times 4 \times 2 = 240$ training examples. The number of output classes for INS 13 is 31 (30 queries plus the background class).

The original Faster R-CNN training parameters described in [167] are kept for fine-tuning, except for the number of iterations, which we decreased to 5.000 considering our small number of training samples. We train a separate network for each one of the tested datasets, using the two different fine-tuning modalities described in Section 3.3. Fine-tuning was performed on a Nvidia Titan X GPU and took around 30 and 45 minutes for finetuning strategies FT_{fc} and FT_{all} , respectively.

The resulting fine-tuned networks are used to extract better image and region representations as well as performing spatial reranking based on class scores instead of feature similarities.

Image Retrieval Pipeline

The three stages of the proposed instance retrieval pipeline are described in this section: filtering stage, spatial reranking and query expansion.

Filtering Stage. The Image-wise pooling (IPA) strategy is used to build image descriptors for both query and database images. At test time, the descriptor of the query image is compared to all the elements in the database, which are then ranked based on the cosine similarity. At this stage, the whole image is considered as the query.

Spatial Reranking. After the Filtering Stage, the top K elements are locally analyzed and reranked. We explore two reranking strategies:

- *Class-Agnostic Spatial Reranking (CA-SR).* For every image in the top K ranking, the region-wise descriptors (RPA) for the top 300 RPN proposals are compared to the region descriptor of the query bounding box. The region-wise descriptors of RPN proposals are pooled from the RoI pooling layer of Faster R-CNN (see Figure 3.1). To obtain the region-wise descriptor of the query object, we warp its bounding box to the size of the feature maps in the last convolutional layer and pool the activations within its area. The region with maximum cosine similarity for every image in the top N ranking gives the object localization, and its score is kept for ranking.
- *Class-Specific Spatial Reranking (CS-SR).* Using a network that has been fine-tuned with the same instances one wishes to retrieve, it is possible to use the direct classification scores for each RPN proposal as the similarity score to the query object. Similarly to CA-SR, the region with maximum score is kept for visualization, and the score is used to rank the image list.

Query Expansion (QE). The image descriptors of the top N elements of the ranking are averaged together with the query descriptor to perform a new search. See Section 2.2 in the previous chapter for further details.

Experiments

This section presents the implementation details of our approach, ablation studies of each of the described components, and comparison with current state of the art methods. Finally, we describe the details of the submission to TRECVID Instance Search 2015, which used the model described in Section 3.3 in the reranking stage.

Experimental Setup

We use both the VGG16 [182] and ZF [222] architectures of Faster R-CNN to extract image and region features. In both cases, we use the last convolutional layer (*conv5* and *conv5_3* for ZF and VGG16, respectively) to build the image descriptors introduced in Section 3.2, which are of dimension 256 and 512 for the ZF and VGG16 architectures, respectively. Region-wise features are pooled from the RoI pooling layer of Faster R-CNN. Images are re-scaled such that their shortest side is 600 pixels. All experiments were run in an Nvidia Titan X GPU.

Off-the-shelf Faster R-CNN features

In this section, we assess the performance of using off-the-shelf features from the Faster R-CNN network for instance retrieval.

First, we compare the sum and max pooling strategies of image- and region-wise descriptors. Table 3.1 summarizes the results. According to our experiments, sum pooling is significantly superior to max pooling for the filtering stage. Such behaviour is consistent

with other works in the literature [11, 93]. Sum pooling is, however, consistently outperformed by max pooling when reranking using region-wise features for all three datasets. Specifically for the Oxford and Paris datasets, we find the spatial reranking with max pooling to be beneficial after filtering (gain of 0.10 and 0.03 mAP points for Oxford and Paris, respectively). However, the spatial reranking (either with max or sum pooling) has little or no effect for the INS13 dataset. To further interpret these results, we qualitatively evaluate the two pooling strategies. Figure 3.2 shows examples of top rankings for INS13 queries, spatially reranked with region-wise max and sum pooled descriptors. These examples indicate that, although mAP is similar, the object locations obtained with max pooling are more accurate. According to this analysis, we set IPA-sum descriptors for the filtering stage and RPA-max descriptors for the spatial reranking in all the upcoming experiments of this chapter.

Table 3.2 shows the performance of different Faster R-CNN architectures (ZF and VGG16) trained on two datasets (Pascal VOC and COCO [116]), including experiments with query expansion with the $N = 5$ top retrieved images as well. As expected, features pooled from the deeper VGG16 network perform better in most cases, which is consistent with previous works in the literature showing that features from deeper networks reach better performance. Query expansion applied after the spatial reranking achieves significant gains for all tested datasets. Such behaviour was expected in particular with Oxford and Paris datasets, for which the spatial reranking already provided a significant gain. Interestingly, query expansion is also most beneficial after spatial reranking for the INS13 dataset, which suggests that, although in this case the spatial reranking does not provide any gain in mAP, the images that fall on the very top of the ranking are more useful to expand the query than the ones in the top of the first ranking.



Figure 3.2: Sum/max comparison on TRECVID. Top 4 rankings and object locations obtained for queries 9098: a *P* (parking automat) sign and 9076: this monochrome bust of Queen Victoria from the INS 2013 dataset (query images surrounded in blue). Comparison between the rankings generated using RPA-sum (top) and RPA-max (bottom), after the filtering stage with IPA-sum. Regressed bounding box coordinates have been disabled for visualization.

Fine-tuned Faster R-CNN

In this section, we assess the impact in retrieval performance of fine-tuning a pretrained network with the query objects to be retrieved. We choose to fine-tune the VGG16 Faster R-CNN model, pretrained with the objects of the Microsoft COCO dataset.

We first take the networks fine-tuned with strategy FT_{fc} and run the retrieval pipeline from scratch. Table 3.3 shows the obtained results (FT_{fc} columns). Results of the filtering and CA-SR stages are the same as those obtained with the original Faster R-

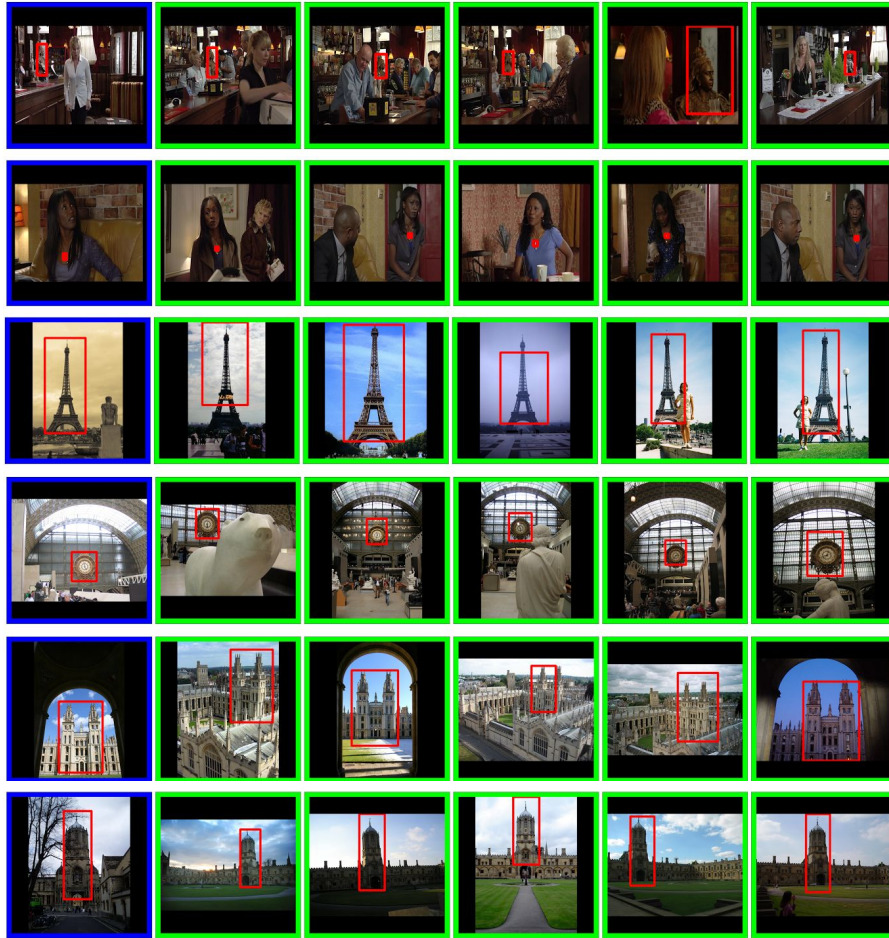


Figure 3.3: Ranking examples and object locations obtained by our proposed retrieval system for query objects (left, depicted with a blue contour) of three different datasets: TRECVID INS 2013, Paris Buildings and Oxford Buildings.

CNN model, which is because the weights for the convolutional layers were not modified during fine-tuning. Results indicate that, although mAP is not always improved after CS-SR (e.g., from 0.588 to 0.543 for Oxford 5k), it is significantly better than CA-SR for Oxford and Paris when followed with query expansion. In case of the INS 13 dataset, we do not find significant improvements when using CS-SR, which suggests that only fine-tuning fully connected layers might not be sufficient to effectively detect the challenging query objects in this dataset.

The second experiment in this section involves fine-tuning a higher number of layers in the Faster R-CNN architecture (Fine-tuning Strategy FT_{all}). Using this modality, the weights in the last convolutional layer are modified. Figure 3.4 shows the difference in the activations in conv5.3 after fine-tuning it for the query instances in each dataset. These visualizations indicate that, after fine-tuning, more neurons in the convolutional layer positively react to the visual patterns that are present in the query objects of the dataset.

We then use the fine-tuned networks of the Fine-tuning Strategy FT_{all} for each one of the datasets to extract image- and region-wise descriptors to perform instance search. Table

Filtering	Reranking	Oxford 5k	Paris 6k	INS 13
	None	0.505	0.612	0.215
IPA-sum	RPA-sum	0.501	0.621	0.196
	RPA-max	0.602	0.641	0.206
	None	0.478	0.540	0.131
IPA-max	RPA-sum	0.508	0.565	0.135
	RPA-max	0.559	0.561	0.138

Table 3.1: Sum/max pooling comparison. Mean Average Precision (mAP) comparison between sum and max pooling strategies for both filtering and reranking stages using *conv5* features from the ZF Faster R-CNN model.

ConvNet	CA-SR	QE	Oxford 5k	Paris 6k	INS 13
ZF (VOC)	No	No	0.505	0.612	0.215
		Yes	0.515	0.671	0.246
	Yes	No	0.602	0.640	0.206
		Yes	0.622	0.707	0.261
VGG16 (VOC)	No	No	0.588	0.657	0.172
		Yes	0.614	0.706	0.201
	Yes	No	0.641	0.683	0.171
		Yes	0.679	0.729	0.242
VGG16 (COCO)	No	No	0.588	0.656	0.216
		Yes	0.600	0.695	0.250
	Yes	No	0.573	0.663	0.192
		Yes	0.647	0.732	0.241

Table 3.2: mAP of pre-trained Faster R-CNN models with ZF and VGG16 architectures. In all cases, IPA-sum descriptors are used for the filtering stage. The CA-SR column specifies whether Class-Agnostic Spatial Reranking with RPA-max is applied to the top $K = 100$ elements of the ranking. When indicated, QE is applied with $N = 5$.

3.3 presents the results (FT_{all} columns). As expected, fine-tuned features significantly outperform raw Faster R-CNN features for all datasets (mAP is $\sim 20\%$ higher for Oxford and Paris, and 8% higher for INS 13). Results indicate that, for Oxford and Paris datasets, the gain of CA-SR + QE is higher with raw features (10% and 11% mAP increase for Oxford and Paris, respectively) than with fine-tuned ones (8% and 3% mAP increase, respectively). This suggests that fine-tuned features are already discriminant enough to correctly retrieve the objects in these two datasets. However, results for the INS 13 dataset show that CA-SR + QE is most beneficial when using fine-tuned features (11% and 41% mAP increase for raw and fine-tuned features, respectively). This difference between the performance for Oxford/Paris and INS13 suggests that queries from the latter are more challenging and therefore benefit from fine-tuned features and spatial reranking the most. A similar behaviour is observed for CS-SR which, for Oxford and Paris, is most beneficial when applied to a ranking obtained with raw features. For INS 13, however, the gain is greater when using fine-tuned features. Overall, the performance of reranking + query expansion is higher for CS-SR than CA-SR. Figure 3.3 shows examples of rankings for queries of the three different datasets after applying

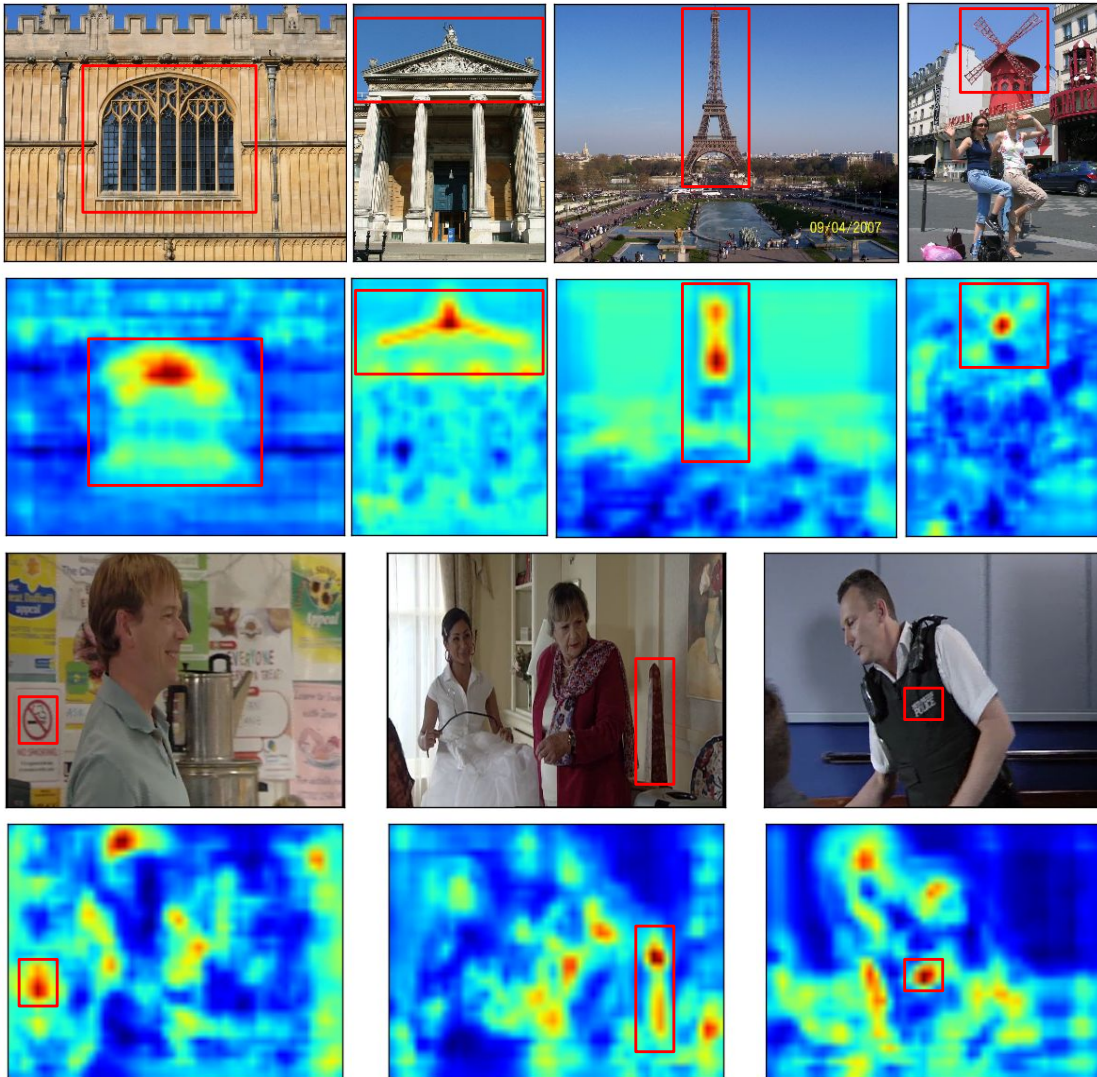


Figure 3.4: Activations after fine-tuning. Difference between *conv5_3* features (sum pooled over feature maps) extracted from the original Faster R-CNN model pretrained with MS COCO with *conv5_3* features from the same model fine-tuned for INS13 (bottom), Oxford and Paris (top) queries.

CS-SR. For visualization, we disable the regressed bounding box coordinates predicted by Faster R-CNN and choose to display those that are directly returned by the RPN. We find that the locations returned by the regression layer are inaccurate in most cases, which we hypothesize is caused by the lack of training data.

Finally, in Figure 3.5 we qualitatively evaluate the object detections after CS-SR using the fine-tuned strategies FT_{fc} and FT_{all} . The comparison reveals that locations obtained with the latter are more accurate and tight to the objects. The Fine-tuning Strategy FT_{all} allows the RPN layers to adapt to the query objects, which causes the network to produce object proposals that are more suitable for the objects in the test datasets.

R	QE	Oxford 5k		Paris 6k		INS 13	
		FT_{fc}	FT_{all}	FT_{fc}	FT_{all}	FT_{fc}	FT_{all}
No	No	0.588	0.710	0.656	0.798	0.216	0.234
No	Yes	0.600	0.748	0.695	0.813	0.250	0.259
CA-SR	No	0.573	0.739	0.663	0.801	0.192	0.248
CA-SR	Yes	0.647	0.772	0.732	0.824	0.241	0.330
CS-SR	No	0.543	0.751	0.793	0.807	0.181	0.250
CS-SR	Yes	0.678	0.786	0.784	0.842	0.250	0.339

Table 3.3: Comparison of fine-tuning strategies FT_{fc} and FT_{all} on the three datasets. Spatial Reranking (R) is applied to the $N = 100$ top elements of the ranking. QE is performed with $M = 5$.

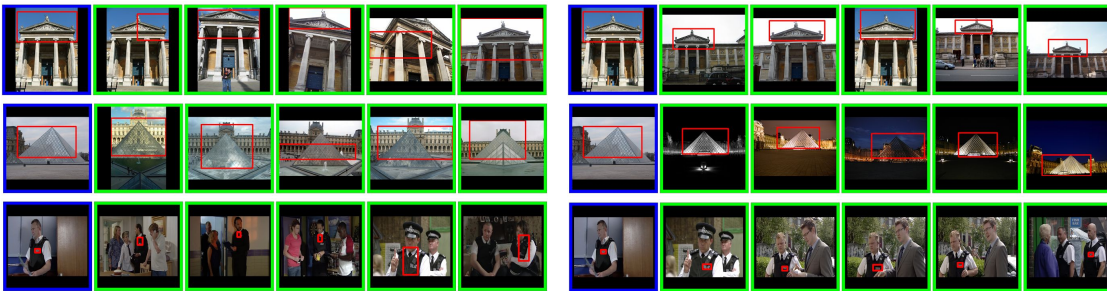


Figure 3.5: Ranking examples after CS-SR with fine-tuned strategies FT_{fc} (left) and FT_{all} (right).

Comparison with state-of-the-art

In this section, we compare our results with several instance search works in the literature. Table 3.4 shows the results of this comparison.

Our proposed pipeline using Faster R-CNN features shows competitive results with respect to the state of the art. However, other works [93, 192] achieve a very high performance without any reranking nor query expansion strategies using similar feature pooling strategies. We hypothesize that the difference in input image size (600px wide in our case vs. full resolution in [93, 192]) can explain the gap in performance. Our proposed reranking strategy CA-SR followed by query expansion is demonstrated to provide similar mAP gains compared to the one proposed in [192]. While CA-SR + QE gives us a gain in mAP of $\sim 10\%$ both for Oxford and Paris (using raw Faster R-CNN features), Tolias et al. [192] use their reranking strategy to raise their mAP by 5 and 15% for the two datasets, respectively.

As expected, results obtained with fine-tuned features (FT_{all}) achieve very competitive results compared to those in the state of the art, which suggests that fine-tuning the network for the object queries is an effective solution when time is not a constraint and query objects are known beforehand. However, the BLCF method introduced in Chapter 2 is able to achieve comparable results with no need of fine tuning the network with query images. That being said, the reranking strategy presented in Chapter 2 provide less significant gains to the performance w.r.t the initial ranking. For Faster R-CNN (FT_{all}), mAP increases from 0.71 to 0.786, and from 0.798 to 0.842 for Oxford and Paris, respectively, while reranking with BLCF boosts mAP from 0.739 to 0.788 and from 0.82

	Oxford 5k	Paris 6k
Razavian et al. [164]	0.556	0.697
R-MAC et al. [192]	0.668	0.830
CroW et al. [93]	0.682	0.796
SPoC [11]	0.657	-
IPA-sum (off-the-shelf)	0.588	0.656
IPA-sum (FT_{all})	0.710	0.798
R-MAC (+ R + QE) [192]	0.770	0.877
CroW (+ QE) [93]	0.722	0.855
IPA-sum (+ IPA-max CA-SR + QE)	0.647	0.732
IPA-sum (FT_{fc}) (+ RPA-max CS-SR + QE)	0.678	0.784
IPA-sum (FT_{all}) (+ RPA-max CS-SR + QE)	0.786	0.842

Table 3.4: Comparison with state-of-the-art works using deep representations instance retrieval.

to 0.848. These results suggest that, while a BoW-based encoding is more powerful than sum-pooling of convolutional features (adopted in this chapter for the filtering stage), a spatial reranking based on features from an object detector trained for the image queries is more effective, and the obtained object locations obtained after reranking are also more accurate (see Figures 2.3 and 3.3 for a qualitative comparison between the two methods). The design choices of the retrieval system will depend on the application requirements. Fine tuning a neural network is prohibitive for generic instance search systems where queries are not known. In these cases, building powerful representations that can encode discriminative information for all possible queries is key to the success of the method. However, results obtained in this chapter indicate that one can reach very similar performance to that of state-of-the art instance search methods by simply training an object detector with a few examples of each query instance.

Finally, the proposed Faster R-CNN fine tuning strategy (FT_{all}) using augmented image queries as training data was used as part of a submission to the TRECVid Instance Search challenge in 2015, in which we obtained the second best result. The approach included an initial ranking stage based on BoW aggregated hand-crafted features, followed by a spatial verification stage relying on the outputs of the fine-tuned Faster R-CNN model.

Conclusion

This chapter has presented different strategies to make use of features from an object detection ConvNet. It provides a simple baseline that uses off-the-shelf Faster R-CNN features to describe both images and their sub-parts. We have shown that is possible to greatly improve the performance of an off-the-shelf based system, at the cost of fine tuning the network for the query images that include objects that one wants to retrieve. The proposed reranking strategy based on fine tuning Faster R-CNN was part of a submission to TRECVid Instance Search 2015 which obtained the 2nd best result in the benchmark.

Summary

This part of the thesis presented two methods based on leveraging features from deep convolutional neural networks for visual instance retrieval. Chapter 2 introduced a BoW-based aggregation of off-the-shelf convolutional features (BLCF), allowing the encoding of the image and its parts for fast retrieval. Chapter 3 presented a retrieval pipeline based on both global and local features extracted from a Faster R-CNN model fine tuned on query images. The two aforementioned strategies achieved competitive results on the Oxford and Paris benchmarks.

Both methods were also evaluated on the challenging TRECVID Instance Search benchmark. In Chapter 2 we demonstrated that BLCF greatly outperforms sum-pooled convolutional features on this benchmark. Chapter 3 presented a reranking based on features from Faster R-CNN fine tuned for query images which improves the results of a strong baseline.

Subsequent to the development of this part of the thesis, several works highlighted the benefits of training ConvNets with ranking-based losses for general-purpose visual instance retrieval [70, 15, 161, 162]. The capabilities of such methodologies will be further explored in Chapter 8 of this thesis, where we learn deep representations for images and text to allow the retrieval of one given the other.

Part II

Image-to-Set Prediction

Introduction

4

Among all image understanding tasks [102, 182, 81, 167, 165, 125, 90, 79, 25], image classification has arguably received most of the attention in computer vision, leading to the development of neural network architectures that reach superhuman performance on some datasets (e.g., the classification test error on the ImageNet dataset [177] is now far below 5% [81]). However, images taken in-the-wild rarely contain a single object, as everyday life pictures are typically complex scenes, which are inherently multi-label. Figure 4.1 shows examples of images in the MS COCO dataset [116] together with their object-level annotations. In contrast to their single-label counterparts, multi-label image annotations present two distinct properties:

- The number of annotation elements is variable across images. In Figure 4.1, the number of category-level labels per image ranges from 1 to 5, while the number of instance-level annotations ranges from 1 to > 20 .
- Annotation elements are unordered, i.e. permuting their positions does not alter their meaning. In the first column in Figure 4.1, the annotation elements for the image can be equivalently represented both as $\{person, skateboard\}$ and $\{skateboard, person\}$.

These two properties make multi-label annotations eligible to be characterized as *sets*. A set is a variable-sized collection of unique elements which is invariant under permutation. In contrast to sequences, sets do not have a predefined order among its items and, while elements in a set can be dependent to one another and exhibit relationships (e.g., a *skateboard* is more likely to appear next to a *person* than to a *horse*), this information

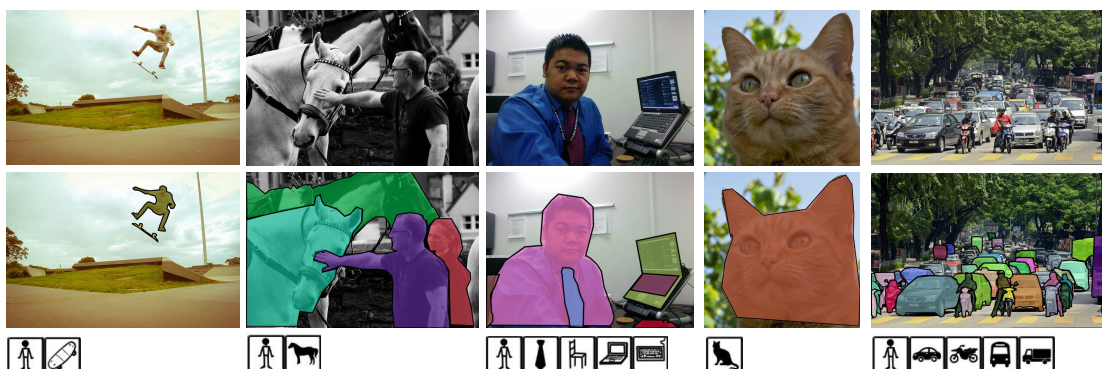


Figure 4.1: Image examples (top) and their object-level annotations (bottom). Samples from MS COCO [116].

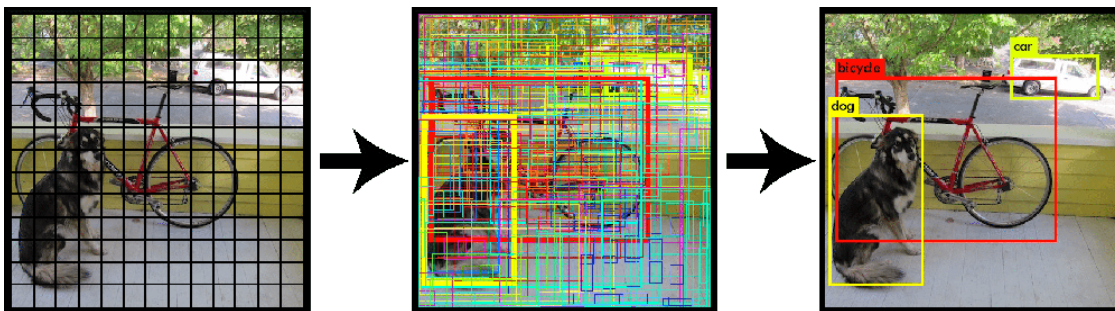


Figure 4.2: NMS for Object Detection. Figure from [165].

is not given explicitly. In computer vision, well-studied problems such as multi-label classification, object detection and instance segmentation can be framed as set prediction tasks, since they require variable-length outputs composed of unordered elements.

Traditional convolutional neural architectures require their outputs to be formulated as fix-sized tensors, e.g., vectors for single-label classification or matrices for pixel-level classification. Since set prediction tasks require variable-sized outputs, most works have tackled them by training deep neural networks with proxy loss functions, whose outputs are then post-processed in order to obtain the final results. In the case of object detection, common methods [64, 63, 167, 165] use convolutional neural networks trained to classify thousands of bounding box locations for the input image. Typically, the number of selected locations is much greater than the actual number of objects that appear in the image, meaning that post-processing is needed to select the set of predictions that better covers all the objects and discard the rest. Although in most recent works the two different stages (i.e. proposal generation and scoring) are optimized jointly [167, 79], the objective function still does not directly model the target task, but a surrogate one which is easier to handle at the cost of an additional filtering step based on non-maximum suppression (NMS), illustrated in Figure 4.2.

While successful, current object detection pipelines present some limitations. First, the number of objects (i.e. the set cardinality) is not explicitly modeled within the network and is estimated in post-processing instead. Although in most cases well optimized, this procedure leads to a waste of computation of scores for thousands of object locations that are eventually discarded. Further, these methods assume that objects in images are independent; although convolutional features are shared for all image locations, each of them is scored independently from its neighbors. Similarly, multi-label classification methods [29, 28, 236] often assume independence among labels in the set, and determine the set cardinality during post-processing (e.g., by thresholding label scores).

Given enough training data and computational power, a great variety of automatic tasks such as object recognition [102], machine translation [190] or speech recognition [73] have seen a boost of performance thanks to models trained end-to-end, i.e. not imposing intermediate representations and directly learning to map the input to the desired output. In this part of the thesis, we explore methods to solve the set prediction task end-to-end, by directly modeling both 1) the likelihood of presence for each element in the set and 2) the set cardinality. In Chapter 5, we extensively evaluate and compare different neural architectures for multi-label classification. Chapter 6 proposes a recurrent neural network to tackle semantic instance segmentation as a set prediction task.

Multi-label Image Classification

5

Multi-label annotation is a go-to standard in many social networks platforms (e.g., hashtags) [130], which equips the research community with large amounts of weak annotations of numerous concepts. Multi-label classification can be naturally framed as an image-to-set prediction problem, since image labels may exhibit relevant dependencies and the number of labels per image is variable. However, many of the existing multi-label classification approaches assume constant set cardinality across images in the dataset [67, 225, 33, 207, 203, 108, 119] and/or are oblivious to label co-occurrences [236, 168, 169]; only a few works model both characteristics of the image-to-set prediction task [112]. Therefore, in this chapter we argue that there is a need to systematically inquire about the importance of modeling co-occurrences among set elements as well as predicting set cardinality.

Another important question is the image-to-set dataset choice, as different datasets are characterized by different levels of label dependencies and different set cardinality distributions. The most widely used datasets for this task are adapted versions of object detection datasets (Pascal VOC [54], MS COCO [116]) or hashtag prediction datasets (NUS-WIDE [38]). Object detection-based datasets contain fully or partially visible object classes exclusively (e.g., dog, table), whereas hashtag prediction datasets may contain classes with higher degree of abstraction (e.g., soccer, party). Moreover, all these datasets have a rather limited number of possible classes (below 100) and a small number of annotations per image (less than 3 on average). Therefore, there is a need to consider more challenging datasets in terms of: (1) class abstraction, (2) number of classes, and (3) number of labels per image. In addition to that, there seems to be no clear consensus among researchers on the metrics to report. On the one hand, some works report performance in terms of mean average precision [208, 209, 218] (reminiscent of object detection) while, on the other hand, others consider intersection-over-union based metrics such as F1 score [67, 112, 225, 33, 32, 207, 203, 236, 168, 169].

The lack of benchmarks with well defined metrics and the rather constrained datasets may hinder the fair comparison between existing methods and slow down advances in the field. Therefore, in this chapter, we argue for a standardized approach to the problem of image-to-set of labels prediction, and present an extensive study of neural network architectures (including feed forward and auto-regressive ones) as well as loss functions (covering binary cross-entropy, soft intersection-over-union, target distribution and cross-entropy) for multi-label classification. We explore different ways of explicitly accounting for class co-occurrences (either through the model architecture or specific loss functions) as well as determining the set cardinality. We compare all tested approaches on five datasets of increasing task complexity, namely Pascal VOC 2007 [54], MS COCO 2014 [116], ADE20k [232] and NUS-WIDE [38] and benchmark all methods in terms of a unified evaluation framework, while ensuring proper and efficient hyperparameter search

through the Hyperband algorithm [109].

The contributions of this chapter can be summarized as:

- We provide an in-depth analysis of the current landscape of image-to-set prediction models, in terms of architectures, loss functions, and their treatment of co-occurrences and set cardinality prediction.
- We evaluate these models using a unified set of metrics on 4 datasets of varying degrees of complexity. Moreover, by carrying extensive hyperparameter tuning for all models, we ensure that differences in performance can be attributed to modeling choices, rather than incomplete hyperparameter optimization.
- Our analysis leads to models that reach state-of-the-art performance on the datasets for which a fair comparison to previous methods is possible.

Our results indicate that auto-regressive models outperform feed-forward ones, with the former models consistently being among the top performers on all of the datasets. This suggests that accounting jointly for both co-occurrences and set cardinality is beneficial. Interestingly, a simple feed-forward network trained with binary cross-entropy loss is also a reasonably good performer in most datasets, reinforcing the importance of proper hyperparameter tuning of baseline models.

Related Work

Multi-label classification has been a long lasting problem in computer vision [226, 85, 230, 212]. Traditionally, the problem has been tackled from many different perspectives, from decomposing the problem into independent binary predictions [146, 226] or modeling label correlations [4, 181, 122], to exploiting priors such as label noise and sparsity [85, 94, 188, 212, 230, 17]. More recently, significant effort has been devoted to leveraging deep neural networks for multi-label classification. Approaches in the deep learning realm often use pre-trained (single-label) image classification models (such as VGG [182] or ResNet [81]) as image feature extractors. Then, they decompose the multi-label classification problem into independent single-label classification problems, by either independently classifying features extracted locally from object proposals [218, 209, 123] or by considering global image features and finetuning the pre-trained models with a binary logistic loss [29, 28, 236]. By considering object proposals separately, the former approaches fail to consider potentially relevant object co-occurrence information. However, the latter approaches could implicitly exploit object co-occurrences from the image global features when deciding on each individual class. Yet, by using a per-class binary logistic loss, these models inherently assume independence among labels.

In order to explicitly capture label co-occurrences, powersets [193] and methods learning the joint probability distribution of labels have been introduced in the literature. Although effective, such methods consider all possible label combinations, and thus can quickly become intractable. To overcome the scalability shortcoming while still modeling label co-occurrences, probabilistic classifier chains [48] and recurrent neural network-based approaches [203, 147, 108, 119] decompose the joint distribution into conditionals at the expense of introducing intrinsic label ordering during training. Therefore,

Model	#outputs	Loss	Dependencies	Cardinality
FF	1	BCE	-	prob. th
FF	1	sIoU	\mathcal{L}	prob. th
FF	1	TD	\mathcal{L}	cum. prob. th
FF	2	BCE	-	DC dist.
FF	2	BCE	-	C dist.
FF	2	sIoU	\mathcal{L}	C dist.
FF	2	TD	\mathcal{L}	C dist.
LSTM	\hat{K}	CE	θ	<i>eos</i> token
LSTM _{set}	\hat{K}	BCE	θ	<i>eos</i> token
TF	\hat{K}	CE	θ	<i>eos</i> token
TF _{set}	\hat{K}	BCE	θ	<i>eos</i> token

Table 5.1: Models summary. Loss-based modeling of label co-occurrences is denoted with \mathcal{L} , while explicitly modeling dependencies in the architecture is represented by θ . Notation: FF (feed-forward), LSTM (long short-term memory), TF (transformer), BCE (binary cross-entropy), sIoU (soft intersection-over-union), TD (target distribution), CE (categorical cross-entropy), DC dist. (Dirichlet-Categorical) and C dist. (Categorical distribution.)

recent works propose to train recurrent neural network-based models either by applying a category-wise max-pooling across the time dimension prior to computing the loss [207, 33, 225] or by optimizing for the most likely ground truth label at each time step [32], effectively getting rid of any enforced order. Other solutions to capture label co-occurrences include learning joint input and label embeddings with ranking-based losses [210, 117, 220, 112, 67] as well as designing loss functions such as target distribution mean squared error [208] or target distribution cross-entropy [67, 130], which directly account for those.

Finally, most state-of-the-art methods are not concerned with estimating the number of labels to be predicted (set cardinality). Instead, they care about evaluating their top- k predictions [67, 225, 33, 207, 203], by manually fixing k for all samples, or apply a fixed threshold to label probabilities [236, 29] (allowing for different number of images per sample). Only recently, multi-label classification has been explicitly addressed as a set prediction problem, where both labels and cardinality are predicted. This is the case of [168, 169, 112], which model set cardinality as a categorical distribution and [112], which learn class-specific probability thresholds.

Image-to-Set Prediction Methods

In image-to-set prediction, we are given a dataset of image and set of labels pairs, with the goal of learning to produce the correct set of labels given an image. The set of labels is an unordered collection of unique elements, which may have variable size. Let $\mathcal{D} = \{d_i\}_{i=1}^N$ be a dictionary of labels of size N , from which we can obtain the set of labels S for an image \mathbf{x} by selecting $K \geq 0$ elements from \mathcal{D} . If $K = 0$, no elements are selected and $S = \{\}$; otherwise $S = \{s_i\}_{i=1}^K$. Thus, our training data consists of M image and label pairs $\{(\mathbf{x}^{(i)}, S^{(i)})\}_{i=1}^M$.

Table 5.1 gives an overview of the image-to-set prediction models considered in this study. A comprehensive overview of set prediction models is out of the scope of this work; we limit the scope of our study to approaches based on feed forward (FF) architectures as well as auto-regressive ones, since they are currently the state-of-the-art for this task. Overall, the models we consider can be categorized according to: (1) whether they model co-occurrences of elements in the set, and (2) whether they explicitly model set cardinality. All models are composed of an image representation module, followed by a set prediction module, which are stacked together and trained end-to-end.

Image Representation. We choose ResNet-50 [81] as image encoder, initialized with pre-trained ImageNet [177] weights, given its ubiquitous role in the literature. The encoder transforms an input image $\mathbf{x} \in \mathbb{R}^{W \times H \times 3}$ into a representation $\mathbf{r} = f_\phi(\mathbf{x})$ of dimensions $w \times h \times 2048$, where w and h are the width and height of the convolutional features, respectively.

Set Prediction. In this work, we consider feed-forward and auto-regressive architectures for image-to-set prediction, which are described in the following subsections.

Feed-forward Models

Notation: We represent S as a binary vector \mathbf{s} of dimension N , where $\mathbf{s}_i = 1$ if $\mathbf{s}_i \in S$ and 0 otherwise¹. The goal is to estimate the label probabilities $\hat{\mathbf{s}}$ from an image \mathbf{x} . Training data consists of M image and set pairs.

Architectures: Feed-forward models take image features \mathbf{r} as input and output $\hat{\mathbf{s}} = g_\theta(\mathbf{r})$. These models are composed of (1) an optional 1×1 convolutional block to change the feature dimensionality of the input features, (2) a global average pooling operation to collapse the spatial dimensions, and (3) one or more fully connected layers. Intermediate fully connected layers are followed by dropout, batch normalization and a ReLU non-linearity. The last fully connected layer serves as classifier, and thus, is followed by a sigmoid non-linearity to obtain the vector of estimated probabilities. The architecture used for all feed-forward models is depicted in Figure 5.1(a).

Loss functions: The model’s parameters are trained by maximizing the following objective over the dataset:

$$\arg \max_{\phi, \theta} \sum_{i=0}^M \log p(\hat{\mathbf{s}}^{(i)} = \mathbf{s}^{(i)} | \mathbf{x}^{(i)}; \phi, \theta). \quad (5.1)$$

where ϕ and θ are the image representation and set predictor parameters, respectively. Most state-of-the-art feed-forward methods assume independence among labels, factorizing $p(\hat{\mathbf{s}}^{(i)} = \mathbf{s}^{(i)} | \mathbf{x}^{(i)})$ as $\sum_{j=0}^N \log p(\hat{\mathbf{s}}_j^{(i)} = \mathbf{s}_j^{(i)} | \mathbf{x}^{(i)})$ and using binary cross-entropy (BCE) as training loss. However, the elements in the set are not necessarily independent. Therefore, we can borrow from the semantic segmentation literature and train the feed-forward set predictor with a soft structured prediction loss, such as the soft intersection-over-union (sIoU), in order to take into account dependencies among elements in the set: Alternatively, we can use the *target distribution* $p(\mathbf{s}^{(i)} | \mathbf{x}^{(i)}) = \mathbf{s}^{(i)} / \sum_j \mathbf{s}_j^{(i)}$ [67, 130] to model the joint distribution of set elements and train a model by minimizing the cross-entropy loss between $p(\mathbf{s}^{(i)} | \mathbf{x}^{(i)})$ and the model’s output distribution $p(\hat{\mathbf{s}}^{(i)} | \mathbf{x}^{(i)})$. Hereinafter, we refer

¹Recall that N represents the size of the label dictionary

to the feed-forward model trained with BCE as FF_{BCE} , the one trained with sIoU as FF_{sIoU} , and the one trained with target distribution as FF_{TD} .

Set cardinality: Given the estimated probabilities $\hat{\mathbf{s}}$ obtained with any of the aforementioned approaches, a set of labels \hat{S} must be recovered. For both FF_{BCE} and FF_{sIoU} , one simple solution is to apply a threshold th to $\hat{\mathbf{s}}$, keeping all labels for which $\hat{s}_i \geq th$. Typically, this threshold is set to 0.5. Nonetheless, in the case of the FF_{TD} , we recover the label set by greedily sampling elements from a *cumulative distribution of sorted output probabilities* $p(\hat{\mathbf{s}}^{(i)}|\mathbf{x}^{(i)})$ and stop the sampling once the sum of probabilities of selected elements is above a threshold $th = 0.5$. Alternatively, the set cardinality K may be explicitly predicted by the feed-forward model through a second output $\hat{\mathbf{s}}, \hat{\mathbf{K}} = g_{\theta}(\mathbf{r})$, where $\hat{\mathbf{K}}$ represents the categorical distribution over possible set cardinalities. At inference time, the top- \hat{K} labels with highest probability are included in the predicted set. For completeness, in our experiments we also use a variant of FF_{BCE} where the set cardinality is modeled with Dirichlet-Categorical distribution, following the model described in [169].

Empty set prediction: Images with missing labels (i.e., $S = \{\}$) can be naturally handled by models that assume label independence (e.g., FF_{BCE} and FF_{sIoU} , whose output is a probability distribution for each label). At inference time, the set cardinality is predicted implicitly by applying a threshold value th to each output probability. The set cardinality can be also modeled explicitly (through a second output), where the output of cardinality 0 corresponds to empty set. From the feed-forward models considered, only FF_{TD} cannot handle empty sets, since a vector with all zeros is not a valid (categorical) probability distribution.

Auto-regressive Models

Notation: When using auto-regressive models, we represent S as a binary matrix \mathbf{S} of dimensions $K \times N$.² We set $\mathbf{S}_{i,j} = 1$ if label d_j is selected at i -th position and 0 otherwise (in other words, each row in \mathbf{S} contains the one-hot-code representation of one label).

Architectures: We explore two auto-regressive architectures, namely a Long Short-Term Memory (LSTM) [82] with spatial attention-based model [128] and a transformer-based (TF) one [196]. Both LSTM and TF take image features \mathbf{r} as input and output $\hat{\mathbf{S}} = g_{\theta}(\mathbf{r})$. These models are composed of (1) an optional 1×1 convolutional block to change the feature dimensionality of the input features, and (2) either a single LSTM layer (following [128]) or several transformer layers (following [196]). The output layer of the model is used as classifier and has a softmax non-linearity. These models predict one element of the set at each time-step. The LSTM and Transformer architectures are depicted in Figures 5.1(b) and 5.1(c), respectively.

Loss functions: In this scenario, the goal is to predict $\hat{\mathbf{S}}$ from an image \mathbf{x} by maximizing the following objective.

$$\arg \max_{\phi, \theta} \sum_{i=0}^M \log p(\hat{\mathbf{S}}^{(i)} = \mathbf{S}^{(i)} | \mathbf{x}^{(i)}; \phi, \theta), \quad (5.2)$$

To ensure that labels in $\hat{\mathbf{S}}^{(i)}$ are selected without repetition, we force the pre-activation

²Recall that K defines the set cardinality and N the size of the dictionary of possible labels.

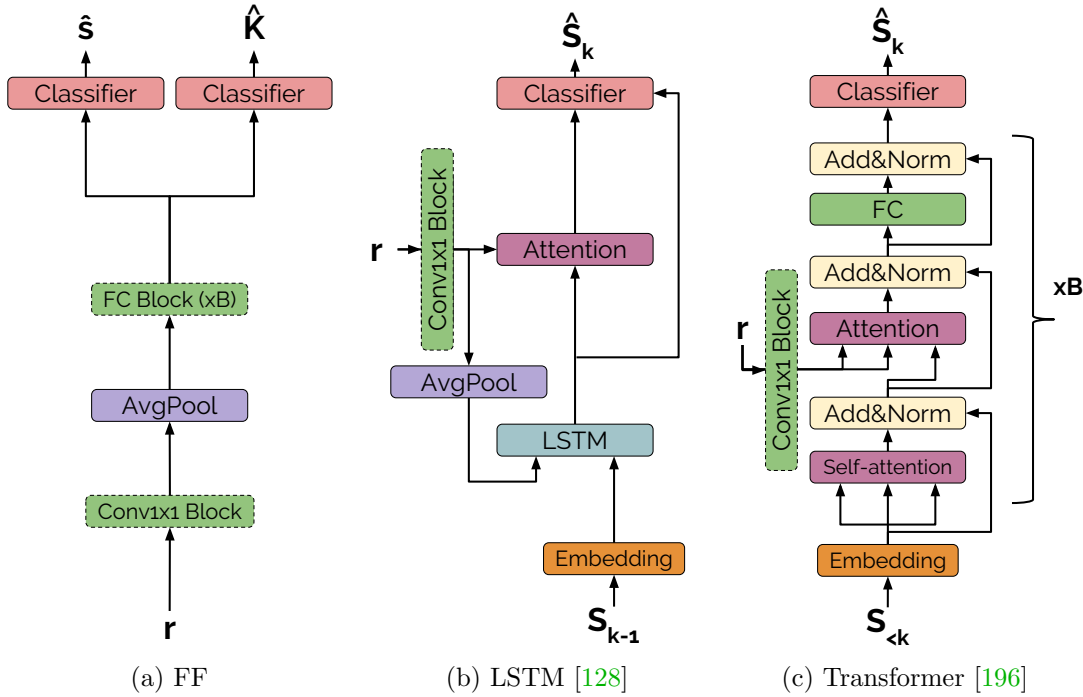


Figure 5.1: Set prediction architectures. (a) Feed-forward (FF), (b) LSTM and (c). Dashed lines denote optional blocks.

of $p(\hat{\mathbf{S}}_k^{(i)} | \mathbf{x}^{(i)}, \mathbf{S}_{<k}^{(i)})$ to be $-\infty$ for all previously selected labels at time-steps $< k$. One characteristic of the formulation in Equation 5.2 is that it inherently penalizes for order, which might not necessarily be relevant for the set prediction task. In order to ignore the order in which labels are predicted, we aggregate the outputs across different time-steps by means of a max pooling operation. In this case, instead of minimizing the cross-entropy error at each time step, we minimize the BCE between the pooled predicted labels and the ground truth. Hereinafter, we refer to the LSTM and TF models trained with pooled time-steps as LSTM_{set} and TF_{set} , respectively. It is worth noting that, in all cases, at inference time, we directly sample from the auto-regressive predictor’s output.

Set cardinality: Most auto-regressive set predictors in the literature are not concerned with cardinality prediction, and predict a fixed number of elements by default [33, 203]. However, we argue that those models inherently have the mechanism to learn when to stop. Therefore, as commonly done in tasks such as image captioning and machine translation, we introduce an end-of-sequence *eos* token to the dictionary of labels, which has to be predicted in the last sequence step. Thus, in our case, the *eos* token’s role is to estimate the cardinality of the set. In the case of LSTM_{set} and TF_{set} , we learn the stopping criteria with an additional loss accounting for it. The *eos* loss is defined as the BCE loss between the predicted *eos* probability at all time-steps and the ground truth (represented as a unit step function, whose value is 0 for the time-steps corresponding to labels and 1 otherwise). In addition to that, we incorporate a cardinality ℓ_1 penalty. In this last case, we weight the contribution of the *eos*-loss and cardinality penalty terms with hyperparameters λ_{eos} and λ_{CP} , respectively.

Empty set prediction: We handle images with missing labels by setting the *eos* token as the first element to be predicted in the sequence.

	VOC	COCO	NUS-WIDE	ADE20k
Train	4 509	74 503	145 610	18 176
Val	502	8 280	16 179	2 020
Test	4 952	40 504	107 859	2 000
N	20	80	81	150
K	1.57 ± 0.77	2.91 ± 1.84	1.86 ± 1.71	8.17 ± 4.14

Table 5.2: Splits, dictionary size (N), and cardinality (K), reported as *mean* (\pm *std*) for each dataset.

Experiments

Datasets and Metrics

We train and evaluate our models on five different image datasets, which provide multi-label annotations. The dataset details are presented in Table 5.2, while the distribution of the training set cardinality is depicted in Figure 5.2.

Pascal VOC 2007 [54] is a popular benchmark for image classification, object detection and segmentation tasks. It is composed of 9 963 images containing objects from 20 distinct categories. Images are divided in 2 501, 2 510 and 4 952 for train, validation and test splits, respectively. We train with 90% of the *trainval* images, keeping 10% for validation. Models are evaluated on the test set, for which annotations are available.

MS COCO 2014 [116] is a popular benchmark for object detection and segmentation on natural images, containing annotations for objects of 80 different categories. It is composed of 82 783 images for training and 40 504 for validation. Since evaluation on the test set can only be done through the benchmark server, which currently does not support the set prediction task, we use 10% of the training set for validation, and evaluate on the full validation set. Note that in our experiments we include images with no annotations as *empty sets*.

NUS-WIDE [38] is a web image database composed of 161 789 images for training and 107 859 for testing, annotated with 81 unique tags collected from Flickr. While VOC and MS COCO are annotated with visually grounded object tags (e.g., *dog*, *train* or *person*), NUS-WIDE includes a wider variety of tags referring to activities (e.g., *wedding*, *soccer*), scenes (e.g., *snow*, *airport*) and objects (e.g., *car*, *computer*, *dog*). As in COCO, this dataset includes images with *empty sets* annotations.

ADE20k [232] is a scene parsing dataset, containing 20 210 training, 2 000 validation samples, annotated with a dictionary of 150 labels. Since the test set server evaluation is not suited for image to set prediction, we use validation set as a test set and separate a new validation set from the training set. As a result we obtain 18 176, 2 020 and 2 000 images for train, validation and test splits, respectively.

Metrics. We evaluate all methods by means of F1 score calculated per-class (C-F1), per-image (I-F1) and overall (O-F1). Note that O-F1 and C-F1 are also commonly referred to as macro- and micro-F1, respectively.

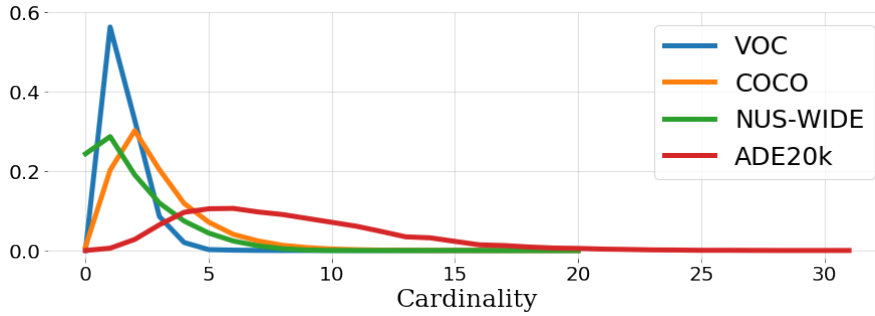


Figure 5.2: Dataset cardinality distribution.

Implementation details

We resize all images to 448 pixels in their shortest side and take random crops of 448×448 for training. We randomly flip ($p = 0.5$), translate (within a range of $\pm 10\%$ of the image size on each axis) and rotate images ($\pm 10^\circ$) for data augmentation during training. All models are trained with the Adam optimizer [99] for a maximum of 200 epochs, or until early-stopping criteria is met (monitoring the O-F1 metric and using patience of 50 epochs for VOC and 10 epochs for the remaining datasets). All models are implemented with PyTorch³ [155]. For autoregressive models, we train on two variants of annotations: (1) we keep the dataset order (e. g. LSTM and TF), and (2) we randomly shuffle the labels each time we load an image (e. g. LSTM_{shuffle} and TF_{shuffle}). For hyperparameter tuning, we allowed Hyperband to sample values from a set of mutually independent categorical distributions, one for each hyperparameter.

Model selection

To tune all model hyperparameters, we used HYPERBAND [109], a bandit-based algorithm that speeds up random search via an aggressive form of early-stopping called SUCCESSIVEHALVING [87]. In SUCCESSIVEHALVING, a set of n different hyperparameter combinations is sampled, each of which is initially allowed to run using r “resources” (e.g., training time, some number of epochs, size used for dataset subsampling). The best n/η of these are kept (according to best O-F1 over validation set observed after using r resources) and subsequently run with ηr resources, where η is a parameter controlling the rate at which values are discarded; this elimination process is repeated until a single best configuration is chosen. However, since SUCCESSIVEHALVING might be too aggressive (i.e., it can discard potentially good configurations in early steps), HYPERBAND hedges by repeating the process multiple times. Each repetition—or “bracket”—uses different hyperparameters n and resource limits r to control the level of aggressiveness; moreover, these values are chosen so that the total resource usage across all runs in each bracket is approximately uniform. This approach has theoretical guarantees that do not rely on strong assumptions about the function to be optimized (in our case best O-F1 over validation set). Moreover, it has been shown to result in substantial computational savings with respect to a random search that does not use SUCCESSIVEHALVING [109]. In our experiments, we used $\eta = 3$, and a maximum value of r equal to $R = 600$, where each resource unit is equivalent to 0.15 training epochs for most datasets, rounding up when necessary (for VOC, equivalent to 0.2 epochs). This translates to roughly 410 hyper-

³<http://pytorch.org/>

Rank	Model	VOC			COCO			NUS-WIDE			ADE20k		
		O-F1	C-F1	I-F1	O-F1	C-F1	I-F1	O-F1	C-F1	I-F1	F1	C-F1	I-F1
1	TF _{shuffle}	86.59 (0.27)	85.48 (0.33)	88.42 (0.28)	77.07 (0.04)	73.72 (0.06)	79.96 (0.03)	68.73 (0.24)	53.62 (0.22)	65.81 (0.48)	70.28 (0.17)	46.11 (0.46)	69.61 (0.15)
2	LSTM	86.33 (0.08)	85.10 (0.14)	88.14 (0.08)	76.66 (0.03)	73.04 (0.03)	79.48 (0.05)	70.54 (0.08)	54.25 (0.17)	67.96 (0.48)	70.50 (0.24)	48.82 (0.58)	69.82 (0.23)
3	LSTM _{shuffle}	87.27 (0.21)	85.75 (0.33)	88.98 (0.17)	77.13 (0.07)	73.61 (0.10)	80.03 (0.07)	67.69 (0.08)	50.19 (0.18)	62.02 (0.11)	69.54 (0.24)	43.23 (0.82)	68.96 (0.29)
4	TF	85.85 (0.18)	84.27 (0.21)	87.84 (0.16)	76.91 (0.05)	73.70 (0.10)	79.68 (0.05)	70.77 (0.03)	55.63 (0.20)	69.41 (0.06)	69.94 (0.24)	47.76 (0.57)	68.86 (0.34)
5	FF _{BCE}	86.57 (0.10)	85.31 (0.12)	88.41 (0.11)	76.56 (0.03)	72.79 (0.11)	78.65 (0.03)	68.87 (0.10)	53.32 (0.15)	56.22 (0.13)	70.15 (0.17)	48.31 (0.31)	68.67 (0.13)
6	LSTM _{set}	86.23 (0.07)	85.26 (0.14)	88.14 (0.08)	76.17 (0.14)	72.78 (0.15)	79.15 (0.09)	69.66 (0.08)	55.74 (0.09)	67.31 (0.06)	70.25 (0.66)	47.47 (2.50)	69.51 (0.56)
7	FF _{BCE,DC}	85.55 (0.43)	83.77 (0.55)	87.69 (0.42)	75.92 (0.04)	71.86 (0.11)	77.98 (0.07)	68.19 (0.07)	52.59 (0.34)	55.38 (0.07)	70.62 (0.18)	46.04 (0.75)	69.90 (0.17)
8	FF _{BCE,C}	84.76 (0.07)	84.22 (0.16)	86.90 (0.08)	69.76 (0.11)	68.10 (0.09)	68.30 (0.12)	61.29 (0.30)	47.25 (0.24)	48.75 (0.17)	70.15 (0.08)	43.94 (0.40)	69.09 (0.07)
9	FF _{TD,C}	84.69 (0.14)	83.52 (0.10)	86.90 (0.11)	70.66 (0.07)	68.81 (0.10)	69.43 (0.07)	63.56 (0.09)	48.19 (0.08)	49.83 (0.09)	69.29 (0.12)	48.57 (0.55)	68.32 (0.17)
10	FF _{sIoU}	87.21 (0.12)	85.97 (0.13)	89.19 (0.08)	73.23 (0.65)	59.96 (1.43)	74.57 (0.76)	62.39 (0.38)	12.85 (0.56)	51.22 (0.17)	67.61 (0.23)	20.79 (0.44)	66.99 (0.24)
11	TF _{set}	86.24 (0.24)	85.18 (0.21)	88.08 (0.23)	52.30 (32.19)	43.94 (38.93)	53.63 (34.34)	57.07 (25.82)	44.16 (23.75)	54.67 (24.98)	50.25 (27.47)	31.22 (25.60)	49.76 (26.90)
12	FF _{sIoU,C}	85.99 (0.23)	84.67 (0.30)	88.11 (0.23)	65.54 (0.50)	52.77 (1.00)	63.80 (0.52)	54.05 (0.74)	9.77 (0.50)	41.53 (0.58)	65.85 (0.21)	20.14 (0.41)	64.97 (0.22)
-	FF _{TD}	79.30 (0.21)	78.50 (0.47)	82.98 (0.21)	-	-	-	-	-	-	63.99 (0.20)	39.47 (0.67)	63.86 (0.24)

Table 5.3: Results on VOC, COCO, NUS-WIDE and ADE20k (test set). We report C-F1, O-F1 and I-F1 computed for each model. Models are trained 5 times using different random seeds. We report *mean (std)* for each metric, model and dataset. The models are ordered according to mean ranking computed over all five tested datasets. Note that FF_{TD} is not considered to obtain the mean ranking, since it is not used for datasets including empty sets (COCO and NUS-WIDE).

parameter configurations evaluated per model, and a maximum budget of 3 200 epochs (4 400 for VOC) for the complete tuning process (with at most 90 training epochs per model); note that we also used patience for monitoring the O-F1 metric during tuning, so this budget is an upper bound. We used the same random seed for all models instantiated during the tuning process.

Analysis

Set label prediction. Table 5.3 reports results for all models and datasets in terms of O-F1, C-F1 and I-F1 metrics. Note that each experiment was run with 5 different seeds (different from the one used for hyper-parameter selection), and thus we report the mean and standard deviation results of each model. Models appear following their average ranking across datasets. According to the ranking, auto-regressive models outperform feed-forward ones. This suggests that explicitly considering both label co-occurrences and set cardinality while training is favorable. Surprisingly, a well tuned very simple baseline (FF_{BCE}) achieves a reasonably high ranking, beating all other feed-forward models.

For VOC dataset, FF_{sIoU} achieves the best performance among feed-forward models, reaching 87.21 O-F1, and closely followed by FF_{BCE} (86.57 O-F1). Interestingly, their

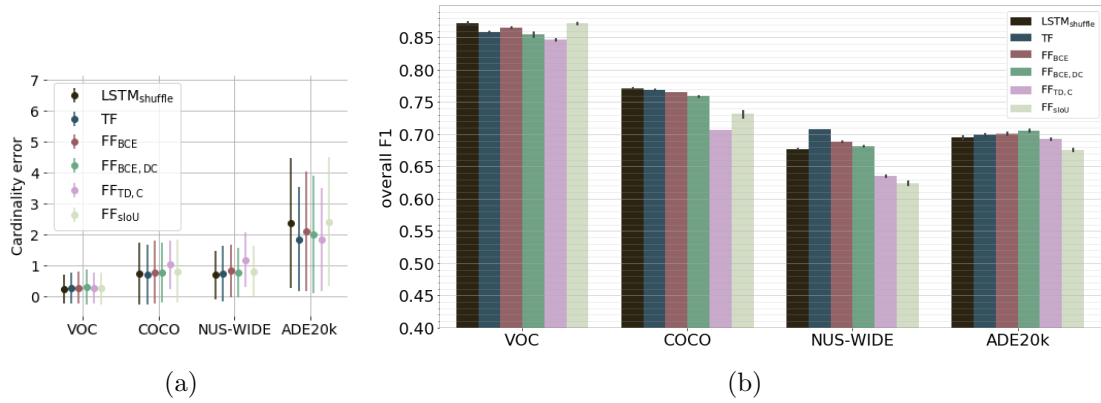


Figure 5.3: (a): Cardinality error. (b): O-F1 per dataset (mean and standard deviation). We compare the best models for each tested dataset as well as two standard feed forward models FF_{BCE} and FF_{sIoU} .

feed-forward counterparts predicting set cardinality achieve slightly worse performance. When it comes to auto-regressive set predictors, $LSTM_{shuffle}$ trained with shuffled labels achieves the best performance with an O-F1 score of 87.27, whereas the rest of these models obtain performances within 1.5 points of $LSTM_{shuffle}$. In the case of COCO dataset, $LSTM_{shuffle}$ is the best performing model (77.13 O-F1), followed by TF and $TF_{shuffle}$. The best feed-forward model for this dataset is FF_{BCE} (76.56 O-F1). Contrary to VOC, auto-regressive models generally outperform feed-forward ones on COCO and, once again, we observe a drop in performance when predicting cardinality in feed-forward models. In the case of NUS-WIDE, auto-regressive models (TF and LSTM) lead the results. It is worth noting that models trained to exploit the dataset order of labels perform better than those trained with shuffled ones. This is not surprising since the label’s order in NUS-WIDE is consistent across all data points (it follows alphabetical order). Similarly to COCO dataset, FF_{BCE} is the best performing feed-forward model, and most of the feed-forward models which predict cardinality are among the least performing ones. When it comes to ADE20k, $FF_{BCE,DC}$ achieves the best performance, with an O-F1 of 70.62 O-F1. In contrast to the previous datasets, endowing feed-forward models with a cardinality prediction path tends to have a rather positive effect. Most auto-regressive models also exhibit good performance in this dataset. While $FF_{BCE,DC}$ and LSTM perform comparably in terms of O-F1, LSTM reaches better C-F1 (48.82 vs 46.04) than $FF_{BCE,DC}$. It is worth mentioning that, as in NUS-WIDE, label order is consistent across samples, and thus can be exploited.

Figure 5.3 presents the test O-F1 metric and cardinality prediction errors for the baseline models FF_{BCE} and FF_{sIoU} as well as the models leading to the best performance for each dataset. As shown in the figure, object detection-based datasets appear to be among the easiest ones, achieving higher overall performance and lower cardinality error, with VOC being the easiest dataset and ADE20k the hardest among them. As for the model architectures, auto-regressive ones seem to be rather consistent across datasets, exhibiting close to top performances and lower cardinality errors. While FF_{IoU} and FF_{BCE} achieve top performance for VOC, these models are outperformed by auto-regressive or feed-forward ones predicting cardinality on all other datasets. This difference can be explained by the higher degree of complexity in COCO, NUS-WIDE and ADE20k compared to VOC (higher dimensionality output space and higher set cardinality). The performance drop

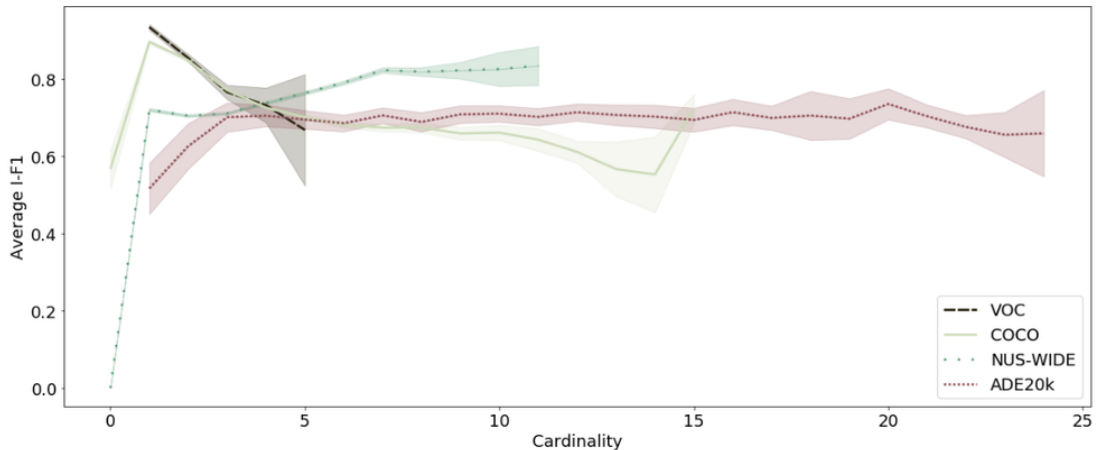


Figure 5.4: I-F1 as a function of cardinality. We report mean I-F1 and 95% confidence intervals for the best models for each dataset at different cardinality values.

can thus be attributed to either a higher set cardinality error or wrong label predictions. Finally, top performing models in terms of O-F1 do not necessarily have lower cardinality prediction error; and similarly, the least performing ones may not correspond to the ones with the highest cardinality error.

Cardinality Prediction. We compare the best models for each dataset in terms of their performance under different set cardinalities in Figure 5.4. The x-axis represents the test set annotation cardinality, while y-axis reports the mean I-F1 that corresponds to each given cardinality value. As shown in the figure, predicting empty sets is hard, e.g., for both COCO and NUS-WIDE, the mean I-F1 is significantly lower for images with cardinality 0 than for images of cardinality 1, a pattern that was consistently observed with other models in these datasets. Moreover, for the datasets that require high level reasoning in order to predict labels, we observe that I-F1 rises with the set cardinality. We hypothesize that this behavior could be attributed to exploiting co-occurrences that improve label predictions (e.g., the more labels we have, the easier it is to predict a label via reasoning about the co-occurrences).

Figure 5.5 shows qualitative results from the best performing model for each dataset.

Comparison to state-of-the-art

In this subsection, we compare our best models to the state-of-the-art. Table 5.4 reports the results in terms of O-F1 for VOC and COCO. Note that state-of-the-art results for NUS-WIDE ignore empty annotations and/or randomly rearrange their splits [32, 236, 112, 119, 123, 108, 225, 67], and thus are not comparable to the results presented in this study. Moreover, to the best of our knowledge, ADE20k has not been used for image-to-set of labels predictions in the past. As shown in the table, we are able to achieve state-of-the-art results in both datasets, even though we challenge our models to predict both the correct labels and set cardinality. This is not the case for the majority of methods evaluated on VOC and COCO. Moreover, a well tuned simple baseline such as FF_{BCE} is able to outperform previous state-of-the art, achieving 86.57 O-F1 on VOC and 76.56 on COCO, showcasing the importance of proper hyperparameter tuning.

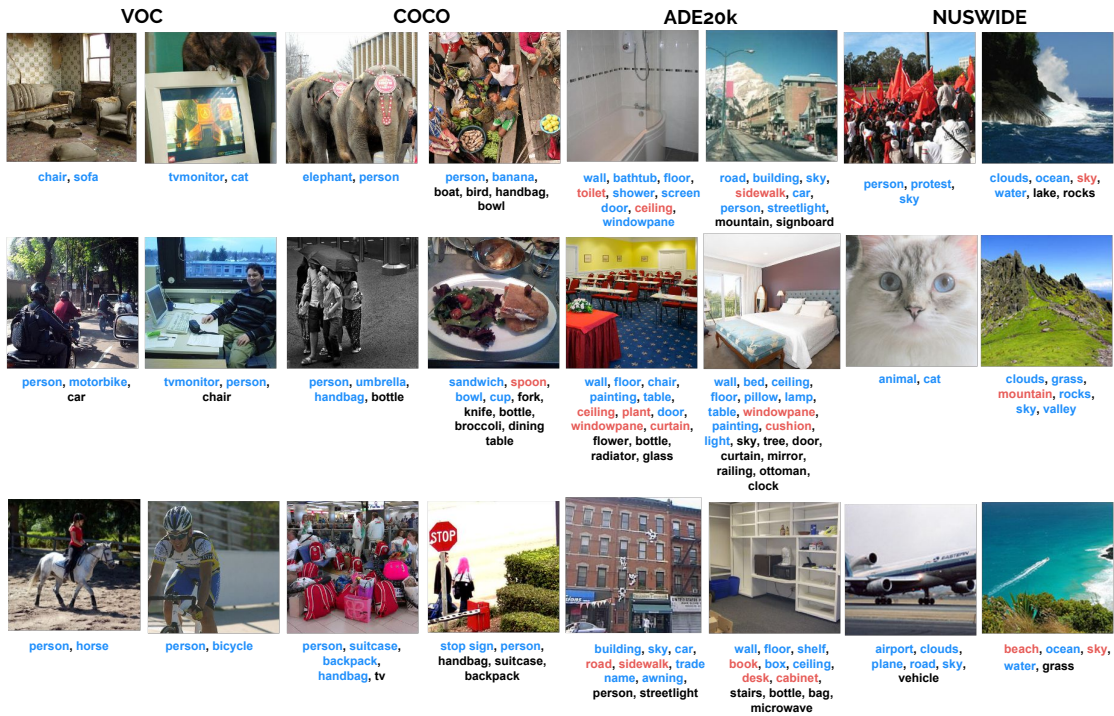


Figure 5.5: Qualitative results. Each column includes two examples for each dataset. True positives, false positives and false negatives are highlighted in blue, red and black, respectively.

	[112]	[236]	[123]	[168]	[169]	[225]	[33]	[32]	[207]	[203]	[108]	[119]	Ours
VOC	79.1	-	-	78.6	81.5	62.9	-	-	-	-	-	-	87.3
COCO	62.9	75.8	74.0	69.0	70.7	66.5	71.1	67.7	72.0	67.8	71.8	75.2	77.1

Table 5.4: Comparison to state-of-the-art on COCO and VOC.

Conclusion

In this chapter, we presented a comprehensive analysis of methods suitable for image-to-set prediction, evaluating their performance in 5 diverse datasets, using a uniform set of metrics and budgets for hyperparameter tuning. Our work reviews the current landscape of image-to-set prediction, and helps elucidate the most promising directions for future research. In particular, our analysis suggests that auto-regressive models are better choices than feed-forward models for the task, performing consistently well across all considered datasets. Moreover, we found that, by exploiting standard ideas of one-to-many sequence models, we can inherently handle set cardinality prediction, label co-occurrences and images without annotations. Additionally, our work emphasizes the importance of thorough hyperparameter tuning, showing that even simple baselines can achieve close to state-of-the-art performance when properly tuned.

Recurrent Instance Segmentation

6

The previous chapter discussed set prediction neural networks for multi-label classification, where each element to be predicted in the set is a categorical label. This chapter takes one step further and presents a set prediction method to solve the task of semantic instance segmentation.

Semantic instance segmentation is defined as the task of assigning a binary mask and a categorical label to each object in an image. In this case, each element in the output set is an object represented by its location in the image (encoded as a binary mask) and its categorical label. Semantic instance segmentation is often understood as an extension of object detection where, instead of bounding boxes, accurate binary masks must be predicted. Both object detection and instance segmentation can be framed as set prediction tasks, since they require variable size outputs containing unordered, yet inter-related elements. Current state of the art methods for semantic instance segmentation [77, 78, 114, 111, 43, 79] extend object detection pipelines based on object proposals [167] by incorporating an additional module that is trained to generate a binary mask for each object proposal. Such architectures follow a two-stage procedure, i.e. a set of object-prominent proposal locations are selected first, and then each of them is given a score, a categorical label and a binary mask.

While most computer vision systems analyze images in a single step, the human exploration of static visual inputs is actually a sequential process [160, 1] that involves reasoning about objects that compose the scene and their relationships. Inspired by this behavior, we design a model that performs a sequential analysis of the scene to deal with complex object distributions and make predictions that are coherent with each other. We take advantage of the capability of Recurrent Neural Networks to generate sequences out of a single input [201, 189] and cast semantic instance segmentation as a sequence prediction task. The model is trained to freely choose the scanpath over the image that maximizes the quality of the segmented instances, which allows us to conduct a detailed study about how it learns to explore images. The object discovery patterns we find are consistent and related to the relative layout of objects in the scene.

Recent works [172, 166] have also proposed sequential solutions for instance segmentation. These are, however, trained to produce a set of class-agnostic masks and must be either evaluated on single-class benchmarks or require a separate method to provide a categorical label for each predicted object. Both [172, 166] impose intermediate representations by using a pre-processed input consisting of a foreground/background mask and instance-level angle information [166] or using an encoder pre-trained for semantic instance segmentation [172]. Based on these works, we develop a true end-to-end recurrent system that provides a set of semantic instances as an output (i.e. both binary masks and categorical labels for all objects in the image) *directly* from image pixels.

The contributions of this work are threefold:

- We present the first end-to-end recurrent model for semantic instance segmentation, trained to predict object instance sets in its output with no required post-processing steps.
- We show its competitive performance against previous sequential methods on three instance segmentation benchmarks, namely Pascal VOC 2012, Cityscapes and CVPPP Plant Leaf Segmentation datasets.
- We provide a thorough analysis of its behavior in terms of the object discovery patterns that it follows once trained.

The remainder of this chapter is structured as follows: Section 6.1 reviews the related work on semantic instance segmentation. Section 6.2 presents our proposed recurrent architecture and describes the training procedure. Section 6.3 discusses the results obtained on single-class and multi-class instance segmentation benchmarks and studies the behavior of the model in terms of the order in which it finds objects. Finally, Section 6.4 draws the conclusions.

Related Work

Most works on semantic instance segmentation inherit their foundations from object detection solutions, augmenting them to segment object proposals [77, 78] and adding post-processing stages to refine the predictions [36]. More recent works build on top of Faster R-CNN [167] by adding a cascade of predictors [43, 42] and iterative refinement of masks [114]. In contrast with cascade-based methods [43, 114, 42], He et al. [79] design an architecture that predicts bounding boxes, segments and class scores in parallel given the output of a fully convolutional network (hence, no chain reliance is imposed). Other works have presented alternative methods to the proposal-based pipelines by treating the image holistically. These include combining object detection and semantic segmentation pipelines with Conditional Random Fields [9], learning a watershed transform on top of a semantic segmentation [14] or clustering object pixels with metric learning [47].

Our model is closer to recent works that formulate the problem of instance segmentation with sequential methods, which predict different object instances one at a time. Ren & Zemel [166] propose a complex multi-task pipeline for instance segmentation that predicts the box coordinates for a different object at each time step using recurrent attention. These bounding boxes are then used to select the image location and predict a binary mask for the object. Their model uses an additional input consisting of a canvas that is composed of the union of the binary masks that have been previously predicted. This architecture resembles two-stage proposal-based ones [77, 114, 79] in the sense that it is also composed of two separate modules, one predicting location coordinates and one to produce a binary mask within this location. The main difference between these works and [166] is that objects are predicted one at a time and are dependent on each other. Romera-Paredes & Torr [172] choose to use a recurrent decoder that stores information about previously found objects in its hidden state. Their model is composed of Convolutional LSTMs [214] that receive features from a pretrained model for semantic segmentation [125] and outputs the separate object segments for the image.

While proposal-based methods have shown impressive performance, they generate an excessive number of predictions and rely on an external post-processing step for filtering them out, e.g., non-maximum suppression. Our proposed recurrent model optimizes an objective which better matches the conditions at inference time, as it is trained to predict the final semantic instance segmentation directly from image pixels. All previous sequential methods [172, 166] are class-agnostic and, although [166] reports results for semantic instance segmentation benchmarks, class probabilities for their predicted segments are obtained from the output of a separate model trained for semantic segmentation. To the best of our knowledge, our proposed method is the first to directly tackle semantic instance segmentation with a fully end-to-end recurrent approach that maps image pixels to a variable length set of objects represented with binary masks and categorical labels.

Model

Given an input image x , the goal of semantic instance segmentation is to provide a set of masks and their corresponding class labels, $y = \{y_1, \dots, y_n\}$. The cardinality of the output set, i.e. the number of instances, depends on the input image and thus the model needs to be able to handle variable length outputs. This poses a challenge for feedforward architectures, which emit outputs of fixed size. Similarly to previous works involving sets [200, 199, 172], we propose a recurrent architecture that outputs a sequence of masks and labels, $\hat{y} = (\hat{y}_1, \dots, \hat{y}_{\hat{n}})$. At any given time step $t \in \{1, \dots, \hat{n}\}$, the prediction is of the form $\hat{y}_t = \{\hat{y}_m, \hat{y}_b, \hat{y}_c, \hat{y}_s\}$, where $\hat{y}_m \in [0, 1]^{h \times w}$ is the binary mask, $\hat{y}_b \in [0, 1]^4$ are the bounding box coordinates normalized by the image dimensions, $\hat{y}_c \in [0, 1]^C$ are the probabilities for the C different categories, and $\hat{y}_s \in [0, 1]$ represents the objectness score, which is the stopping criterion at test time. Obtaining bounding box annotations from the segmentation masks is straightforward and it adds an additional training signal, which resulted in better performing models in our experiments.

We design an encoder-decoder architecture that resembles typical ones from semantic segmentation works [125, 173], where skip connections from the layers in the encoder are used to recover low level features that are helpful to obtain accurate segmentation outputs. The main difference between these works and ours is that our decoder is recurrent, enabling the prediction of one instance at a time instead of a single semantic segmentation map where all objects are present, thus allowing to naturally handle variable length outputs.

Encoder

We use a ResNet-101 [81] model pretrained on ImageNet [177] for image classification as an encoder. We truncate the network at the last convolutional layer, thus removing the last pooling layer and the final classification layer. The encoder takes an RGB image $x \in \mathbb{R}^{h \times w \times 3}$ and extracts features from the different convolutional blocks of the base network $F = \text{encoder}(x)$. F contains the output of each block $F = [f_0, f_1, f_2, f_3, f_4]$, where f_0 corresponds to the output of the deepest block, and f_4 is the output of the block whose input is the image (i.e. $f_{4..0}$ correspond to the output of ResBlock_{1..5} in ResNet-101, respectively).

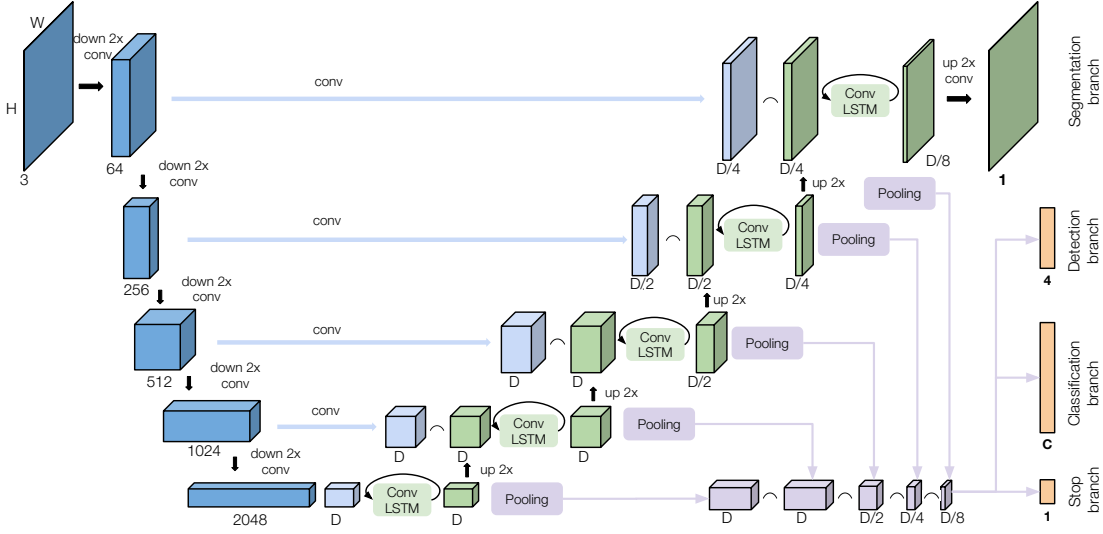


Figure 6.1: Our proposed recurrent architecture for semantic instance segmentation.

Decoder

The decoder receives as input the convolutional features F and outputs a set of \hat{n} predictions, being \hat{n} variable for each input image. Similarly to [172], we use the Convolutional LSTMs [214] as the basic block of our decoder, in order to naturally handle 3-dimensional convolutional features as input and preserve spatial information. A ConvLSTM unit can be written with the following equations:

$$\begin{aligned}
 f_t &= \sigma(x_t * W_f + h_{t-1} * U_f + b_f) \\
 i_t &= \sigma(x_t * W_i + h_{t-1} * U_i + b_i) \\
 o_t &= \sigma(x_t * W_o + h_{t-1} * U_o + b_o) \\
 c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(x_t * W_c + h_{t-1} * U_c + b_c) \\
 h_t &= o_t \odot \tanh(c_t)
 \end{aligned} \tag{6.1}$$

where σ is the sigmoid activation, \odot represents the point-wise multiplication and $*$ represents the convolutional operator. i_t, f_t, o_t are the input, forget and output gates, c_t is the cell state and h_t is the hidden state. W and U terms are convolutional kernels and b represents the bias term.

While [172] uses a two-layer Convolutional LSTM module that receives the output of the last layer of their encoder, we design a hierarchical recurrent architecture that can leverage features from the encoder at different abstraction levels. We design an upsampling network composed of a series of ConvLSTM layers, whose outputs are subsequently merged with the side outputs F from the encoder. This merging can be seen as a form of skip connection that bypasses the previous recurrent layers. Such architecture allows the decoder to reuse low level features from the encoder to refine the final segmentation. Additionally, since we are using a recurrent decoder, the reliance on these features can change across different time steps.

The output of the i^{th} ConvLSTM layer in time step t , $h_{i,t}$, depends on both (a) the input

it receives from the encoder and its preceding ConvLSTM layer and (b) its hidden state representation in the previous time step $h_{i,t-1}$:

$$h_{i,t} = \text{ConvLSTM}_i([B_2(h_{i-1,t}) \mid S_i], h_{i,t-1}) \quad (6.2)$$

where B_2 is the bilinear upsampling operator by a factor of 2, $h_{i-1,t}$ is the hidden state of the previous ConvLSTM layer and S_i is the result of projecting f_i to have lower dimensionality via a convolutional layer.

Equation 6.2 is applied in chain for $i \in \{1, \dots, n_b\}$, being n_b the number of convolutional blocks in the encoder ($n_b = 5$ in ResNet). $h_{0,t}$ is obtained by a ConvLSTM with S_0 as input (i.e. no skip connection):

$$h_{0,t} = \text{ConvLSTM}_0(S_0, h_{0,t-1}) \quad (6.3)$$

We set the first two ConvLSTM layers to have dimension D , and set the dimension of the remaining ones to be the one in the previous layer divided by a factor of 2. All ConvLSTM layers use 3×3 kernels which, compared to 1×1 ConvLSTM units used in [172], have a larger receptive field which can model instances that are far apart more easily. Finally, a single-kernel 1×1 convolutional layer with sigmoid activation is used to obtain a binary mask of the same resolution as the input image.

The bounding box, class and stop prediction branches consist of three separate fully connected layers to predict the 4 box coordinates, the category of the segmented object and the objectness score at time step t . These three layers receive the same input h_t , which is obtained by concatenating the max-pooled hidden states of all ConvLSTM layers in the network. Figure 6.1 shows the details of the recurrent decoder for a single time step.

Training

The parameters of our model are estimated by optimizing a multi-task objective composed of four different terms:

Segmentation loss (L_m): similarly to other works [172, 166], we use the soft intersection over union loss (sIoU) as the cost function between the predicted mask \hat{y} and the ground truth mask y , $\text{sIoU}(\hat{y}, y) = 1 - \frac{\langle \hat{y}, y \rangle}{\|\hat{y}\|_1 + \|y\|_1 - \langle \hat{y}, y \rangle}$.

In order to treat the outputs of our model as sets, we do not impose any specific instance order to match the predictions of our model with the objects in the ground truth. Instead, we let the model decide which output permutation is the best and sort the ground truth accordingly¹. We assign a prediction to each of the ground truth masks by means of the Hungarian algorithm, using sIoU as the cost function. Given a sequence of predicted masks $\hat{y}_m = (\hat{y}_{m,1}, \dots, \hat{y}_{m,\hat{n}})$ and the set of ground truth masks $y_m = \{y_{m,1}, \dots, y_{m,n}\}$,

¹We also experimented with forcing the output sequence to follow hand-designed patterns, but it resulted in low-performing models.

the segmentation loss L_m can be expressed as:

$$L_m(\hat{y}_m, y_m, \delta) = \sum_{t=1}^{\hat{n}} \sum_{t'=1}^n sIoU(\hat{y}_{m,t}, y_{m,t'}) \delta_{t,t'} \quad (6.4)$$

where δ is the matrix of assignments. $\delta_{t,t'}$ is 1 when the predicted and ground truth masks $\hat{y}_{m,t}$ and $y_{m,t'}$ are matched and 0 otherwise. In the case where $\hat{n} > n$, gradients for predictions at $t > n$ are ignored.

Classification loss (L_c): our network outputs class probabilities for each of the predicted masks. Given the sequence of class probabilities $\hat{y}_c = (\hat{y}_{c,1}, \dots, \hat{y}_{c,\hat{n}})$ and the set of ground truth one-hot class vectors $y_c = \{y_{c,1}, \dots, y_{c,n}\}$, the classification loss is computed as the categorical cross entropy between the matched pairs determined by δ .

Detection loss (L_b): given the sequence of predicted bounding box coordinates $\hat{y}_b = (\hat{y}_{b,1}, \dots, \hat{y}_{b,\hat{n}})$ and the ground truth $y_b = \{y_{b,1}, \dots, y_{b,n}\}$, the penalty term L_b for bounding box regression is given by the mean squared error between the box coordinates of matched pairs determined by δ .

Stop loss (L_s): the model emits an objectness score at each time step, $\hat{y}_{s,t}$. It is optimized with a loss term defined as the binary cross entropy between $\hat{y}_{s,t}$ and $\mathbb{1}_{t \leq n}$, where n is the number of instances in the image.

The total loss is the weighted sum of the four terms: $L_m + \alpha L_b + \lambda L_c + \gamma L_s$, where loss terms are subsequently added as training progresses. When training for datasets with a high number of objects per image (i.e. Cityscapes and CVPPP) we use curriculum learning [16] to guide the optimization process, where we begin optimizing the model to predict only two objects and increase this value by one once the validation loss plateaus.

Experiments

In this section, we describe the experimental setup, including the datasets and evaluation metrics. We compare our model with other sequential methods for semantic instance segmentation, and provide an analysis of the object order learned by the network.

Datasets and metrics

We evaluate our models on three benchmarks previously used for semantic instance segmentation that differ from each other in terms of the average amount of objects per image. This diversity in datasets will allow assessing our model based on the length of the sequence to be generated.

Pascal VOC 2012 [54] contains objects of 20 different categories and an average of 2.3 objects per image. Despite having a small number of objects on average, images in this dataset are complex and substantially different from each other in terms of the objects spatial arrangement, scale and pose. Following standard practices in [114, 47, 120], we train with the additional annotations from [76] and evaluate on the original validation set, composed of 1,449 images.

CVPPP Plant Leaf Segmentation [141] is a small dataset of images of different

	Rec	Cls	Pascal VOC	CVPPP		Cityscapes			
			$AP_{person,50}$	SBD \uparrow	DiC \downarrow	AP	AP_{50}	AP_{car}	$AP_{car,50}$
[166]	\times	\times	–	84.9 (± 4.8)	0.8 (± 1.0)	9.5	18.9	27.5	41.9
[172]	\checkmark	\times	46.6	56.8(± 8.2)	1.1(± 0.9)	–	–	–	–
[172] + CRF	\checkmark	\times	50.1	66.6(± 8.7)	1.1(± 0.9)	–	–	–	–
Ours	\checkmark	\checkmark	60.7	74.7(± 5.9)	1.1(± 0.9)	7.8	17.0	25.8	45.7

Table 6.1: Comparison against state of the art sequential methods for semantic instance segmentation. We specify whether the method is recurrent (Rec) and produces categorical probabilities (Cls).

plants. We follow the same scheme as in [172, 166], using only 128 images from the A1 subset for training. The number of leaves per image ranges from 11 to 20, with an average of 16.2. Results are evaluated on 33 test images. While the number of objects per image is significantly higher than in Pascal VOC, this dataset only contains objects from a single category and images present structural similarities that facilitate the task.

Cityscapes [40] contains 5,000 street-view images containing objects of 8 different categories. The dataset is split in 2,975 images for training, 500 for validation and 1,525 for testing. There are, on average, 17.5 objects per image in the training set, with the number of objects ranging from 0 to 120. The large number of instances per image makes this dataset particularly challenging for our model.

We resize images to 256×256 pixels for Pascal VOC, 256×512 for Cityscapes and 500×500 for CVPPP. We evaluate the CVPPP dataset with the symmetric best dice (SBD) and the difference in count (DiC) as in [141]. For Cityscapes and Pascal VOC we report the average precision AP at different IoU thresholds.

Experimental setup

We use the Adam optimizer [99] with a learning rate of 10^{-3} for all layers in the decoder, 10^{-6} for the layers in the encoder. We set $D = 128$ for Pascal and Cityscapes, and $D = 64$ for CVPPP.

We train our model by subsequently adding penalty terms to the loss function one at a time as training progresses. In our experiments we observe that while the penalty term for instance classification L_c quickly converges, the task of segmenting and detecting one object at a time is much more challenging to learn. We hypothesize that this is mainly due to the fact that the encoder we use is pretrained for image classification and not segmentation. To facilitate convergence, we first train the network for 20 epochs with the objective: $L_t = L_m + \alpha L_b$ for 20 epochs (α is set to 10) and add the classification penalty afterwards with $\lambda = 0.1$. Similarly, the penalty term L_s also converges quickly, therefore we set it to 0 and activate it after the model converges for $L_t = L_m + 10L_b + 0.1L_c$. At this point we add the stopping loss term L_s to the cost function with $\gamma = 0.5$ for Pascal VOC and Cityscapes and $\gamma = 0.1$ for CVPPP, and resume training until convergence.

We use typical data augmentation strategies during training: we apply a random rotation with a degree in the range $[-10, 10]$ ($[-180, 180]$ for CVPPP), random translation in both horizontal and vertical axes within a range of 10% of the pixel width and height, respectively. We apply a random shear in a range of 10%, we zoom in and out of the

image within a range of $[0.7, 1.5]$ and randomly flip images with a 0.5 probability.

At inference time, we use a threshold of 0.5 to generate binary masks from the mask output of the network after the sigmoid activation, and stop making predictions for an image once the stopping score goes below 0.5 for Pascal VOC and Cityscapes, and 0.3 for CVPPP.

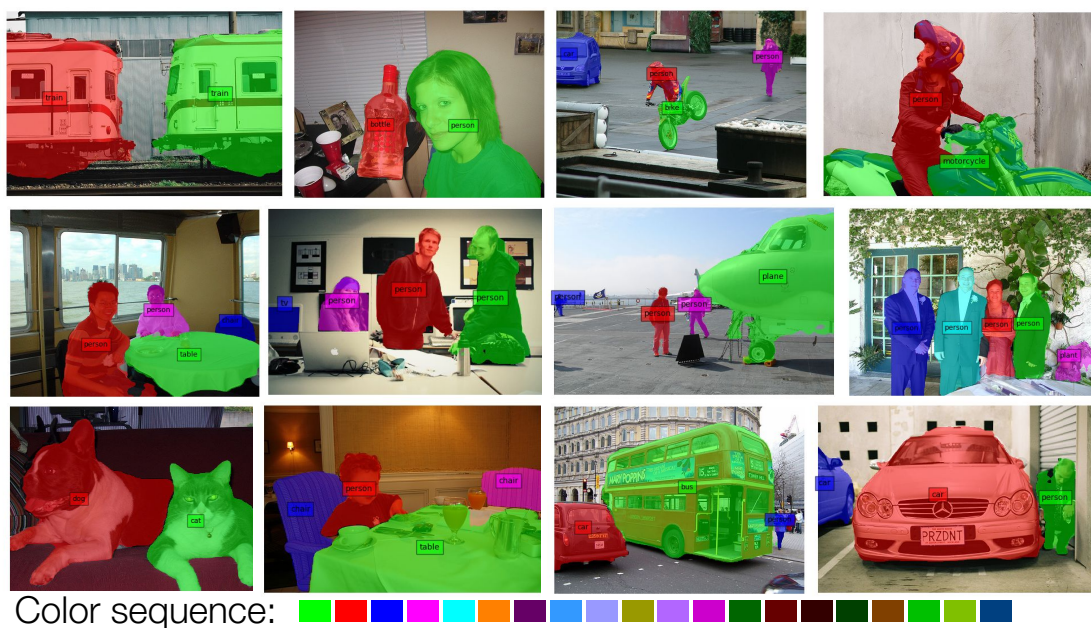
Comparison with sequential methods

We compare our results against other sequential models for instance segmentation [172, 166]. Table 6.1 summarizes the results.

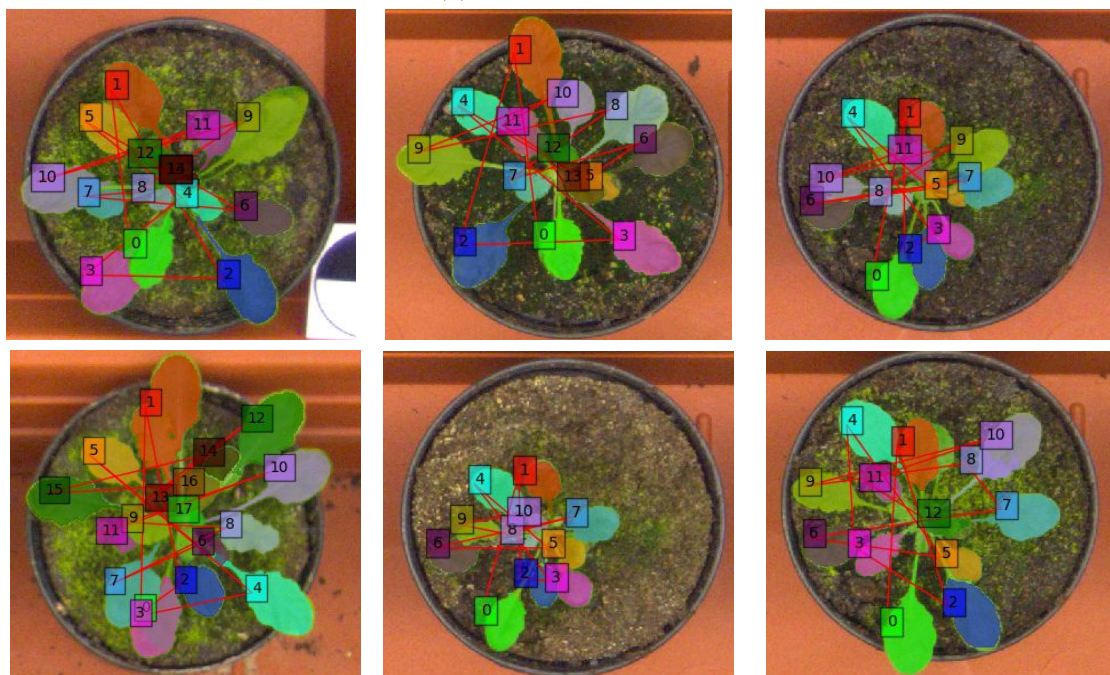
We first train and evaluate our model with the Pascal VOC dataset. In Table 6.1 we compare our method with the recurrent model in [172], whose approach is the most similar to ours. However, since they train and evaluate their method on the person category only, we report the results for this category separately despite that our model is trained for all 20 categories. We outperform their results by a significant margin (AP_{50} of 46.6 vs. 60.7), even in the case in which they use a post processing based on CRFs, reaching an AP_{50} of 50.1. Figure 6.2(a) shows examples of predicted object sequences for Pascal VOC images. Table 6.2b compares our approach with non-sequential methods. We outperform early proposal-based ones [77, 36] by a significant margin across all IoU thresholds. Compared to more recent works [114, 8, 113, 9], our method falls behind for lower thresholds, but remains competitive and even superior in some cases for higher thresholds.

In the case of the CVPPP dataset, our method also outperforms the one in [172] by a significant margin. However, the sequential model in [166] obtains better results in this benchmark. Their method incorporates an input pre-processing stage and involves multi-stage training with different levels of supervision. In contrast with [166], our method directly predicts binary masks from image pixels without imposing any constraints regarding the intermediate feature representation. In Figure 6.2(b) we show examples of predictions obtained by our model for this dataset. Although the number of objects is much higher in this benchmark than in Pascal VOC, our model is able to accurately output one object at a time.

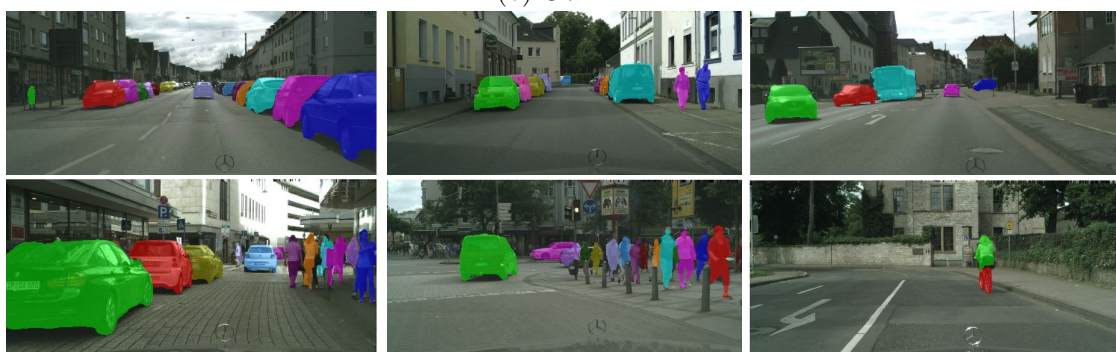
Our performance on Cityscapes is comparable to the results of the only sequential method previously evaluated on this dataset [166], but does not meet state of the art results obtained by non-sequential methods, which reach AP_{50} figures of 58.1 [79], 35.9 [47] and 35.3 [14]. Figure 6.2(c) depicts some sample predictions of our model for this dataset. While our approach is competitive or even better than [166] for simpler and frequent objects (e.g., AP_{50} figures of 45.7 vs. 41.9 for *car*, and 20.5 vs. 21.2 for *person*), it obtains lower scores for less frequent and commonly smaller instances (e.g., 2.8 vs. 10.5 for *bike* and 6.8 vs. 14.7 for *motorbike*). We hypothesize that, as the segmentation module in [166] extracts features at a local scale once the detection module predicts a bounding box, their model can accurately predict binary masks for small instances. In contrast, our method operates at global scale for all instances, generating one binary mask at a time considering all pixels in the image. Working with images at higher resolution would allow us to improve our metrics (specially for small objects), which would come at a cost of higher computational requirements. It is also worth noting that the classification scores in [166] are provided by a separate module trained for the task of semantic segmentation,



(a) Pascal VOC 2012



(b) CVPPP



(c) Cityscapes

Figure 6.2: Examples of generated output sequences for the three datasets.

while our method predicts them together with the binary masks. To the best of our knowledge, ours is the first recurrent model used as a solution for Cityscapes.

Ablation studies

In this section, we quantify the effect of each of the components in our network (encoder, skip-connections and number of recurrent layers). Table 6.2a presents the results of these experiments for Pascal VOC. First, we compare the performance of different image encoders. We find that a deeper encoder yields better performance, with a 23.87% relative increase from VGG-16 to ResNet-101. Further, we analyze the effect of using different skip connection modes (i.e. summation, concatenation and multiplication), as well as removing them completely. While there is little difference between the different skip connection modes, concatenation has better performance. Completely removing skip connections causes a drop of performance of 6.6%, which demonstrates the effectiveness of using them to obtain accurate segmentation masks. We also quantify the effect of reducing the number of ConvLSTM layers in the decoder. To remove ConvLSTM layers, we simply truncate the decoder chain and the output of the last ConvLSTM is upsampled to match the image dimensions. This becomes the input to the last convolutional layer that outputs the final mask. Removing a ConvLSTM layer also means removing the corresponding skip connection. (e.g., if we remove the last ConvLSTM layer, the features from the first convolutional block in the encoder are never used in the decoder). Results in table 6.2a show a decrease in performance as we remove layers from the decoder, which indicates that both the depth of the decoder and the skip connections coming from the encoder contribute to the result. Notably, keeping the original five ConvLSTM layers in the decoder but removing the skip connections provides a similar performance as using a single ConvLSTM layer without skip-connections (AP of 53.3 against 53.2). This indicates that a deeper recurrent module can only improve performance if the side outputs from the encoder are used as additional inputs.

Encoder	skip	N	AP ₅₀	AP _{person,50}		Model	AP ₅₀	AP ₆₀	AP ₇₀	AP ₈₀
VGG16	concat	5	46.5	51.7						
R50	concat	5	53.0	53.9						
R101	concat	5	57.0	60.7						
R101	sum	5	56.7	57.8		SDS [77]	43.8	34.5	21.3	8.7
R101	mult	5	56.1	59.2		Chen et al. [36]	46.3	38.2	27.0	13.5
R101	none	5	53.8	51.3		PFN [113]	58.7	51.3	42.5	31.2
						R2-IOS [114]	66.7	58.1	46.2	–
R101	concat	4	56.0	59.0		Arnab et al. [8]	58.3	52.4	45.4	34.9
R101	concat	3	56.1	59.5		Arnab et al. [9]	61.7	55.5	48.6	39.5
R101	concat	2	54.5	54.0		MPA [120]	60.3	54.6	45.9	34.3
R101	-	1	53.3	50.6		Ours	57.0	51.8	41.5	37.8

(a)

(b)

Table 6.2: Results for Pascal VOC 2012 validation set. (a) Ablation studies. (b) Comparison with the state of the art for different IoU thresholds.

Error analysis

Following standard error diagnosis studies for object detectors [83], we show the distribution of false positive (FP) errors, considering the following types: localization errors

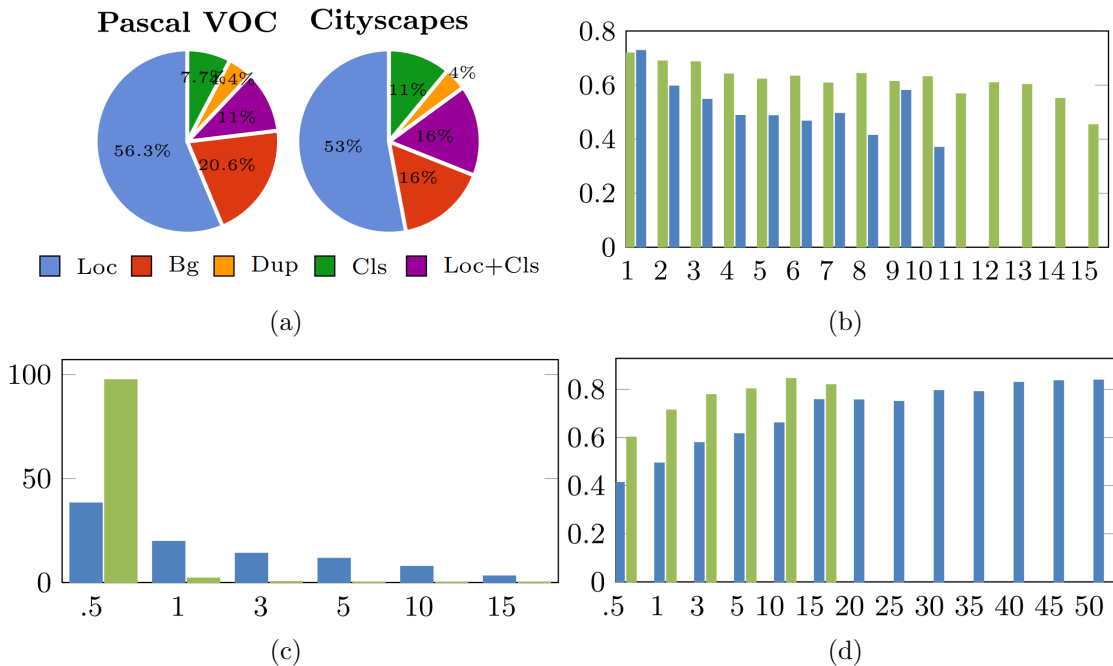


Figure 6.3: (a) False positive distribution. (b-d) Error analysis on Pascal VOC (blue) and Cityscapes (green): (b) IoU vs time step, (c) False negative size distribution, (d) IoU vs object size (object size given as the image % it covers). Reported values in (a) and (d) are constrained to the particularities of each dataset (object sequences for Pascal VOC are shorter and objects in Cityscapes are smaller).

(Loc), confusions with the background (Bg), duplicates (Dup), miss-classifications (Cls), and double localization and classification errors (Loc+Cls). Figure 6.3(a) shows that most FPs are caused by inaccurate localization. Further, in Figure 6.3(b) we show the mask quality in terms of IoU depending on the time step when it was predicted. It can be observed that the quality of the masks degrades as the number of time steps increases. We believe that, as features extracted from the encoder are fixed for any output sequence length, more information has to be encoded in the same feature size for long sequences, acting as a bottleneck. The same applies to the decoder, that must retain more information for longer sequences in order to decide what to output next. These intrinsic properties of a recurrent model may lead to poor mask localization for the last masks of the output prediction. A performance drop for longer sequences when using RNNs has already been demonstrated in other works [13]. Further, we analyze the distribution of false negatives in terms of their size with respect to the image dimensions. We cluster objects in different bins according to the image percentage they cover. Figure 6.3(c) shows that, for both datasets, most of the false negatives (97% and 38% for Cityscapes and Pascal VOC, respectively) are small objects that cover less than 1% of the image. Figure 6.3(d) shows the average IoU for objects of different sizes. Both figures indicate that our method achieves higher IoU values for big objects and struggles with small ones.

Object Sorting Patterns

We observe that the outputs of the model follow a consistent order across images in CVPPP, as depicted in Figure 6.2(b). The complexity and scale of Pascal VOC and

Cityscapes make this qualitative analysis unfeasible, so we analyze the sorting patterns learned by the network by computing their correlation with three predefined sorting strategies: right to left (*r2l*), bottom to top (*b2t*) and large to small (*l2s*). We take the center of mass of each object to represent its location and its area as the measure for its size.

We sort the sequence of predicted masks according to one of the strategies and compare the resulting permutation indices with the original ones using the Kendall tau correlation metric: $\tau = \frac{P-Q}{N(N-1)/2}$. Given a sequence of masks $x \in (x_1, \dots, x_N)$ and its permutation $y \in (y_1, \dots, y_N)$, P is the number of concordant pairs (i.e. pairs that appear in the same order in the two lists) and Q is the number of discordant pairs. $\tau \in [-1, 1]$, where 1 indicates complete correlation, -1 inverse correlation and 0 means there is no correlation between sequences. Table 6.3a presents the results for this experiment. For simplicity, we do not show the results for the opposite sorting criteria in the table (i.e. left to right, small to large and top to bottom), since their τ value would be the same but with the opposite sign. We observe strong correlation with a horizontal sorting strategy for both datasets (right to left in Pascal VOC and left to right in Cityscapes), as well as with bottom to top and large to small patterns.

			Pascal VOC		CVPPP		Cityscapes		
			before	after	before	after	before	after	
	Pascal VOC	Cityscapes							
r2l	0.4916	-0.4428	f_4	-0.048	-0.062	-0.129	0.232	-0.127	-0.162
b2t	0.2788	0.2712	f_3	0.014	-0.005	0.032	0.135	0.279	0.194
l2s	0.2739	0.1700	f_2	-0.088	-0.125	-0.317	-0.141	-0.111	0.144
			f_1	0.008	0.286	0.184	0.505	0.010	0.188
			f_0	0.274	0.634	-0.054	0.147	-0.125	0.209

(a)

(b)

Table 6.3: Analysis of object sorting patterns. Correlation values are given by the Kendall tau coefficient τ . (a) Correlation with predefined patterns. (b) Correlation with convolutional activations. $f_{4..0}$ correspond to the output of ResBlock_{1..5} in ResNet-101, respectively.

Figure 6.4 shows images in Pascal VOC that present high correlation with each of the three sorting strategies. Interestingly, the model adapts its scanning pattern based on the image contents, choosing to start from one side when objects are next to each other, or starting from the largest one when the remaining objects are much smaller. The pattern in Cityscapes is more consistent, which we attribute to the similar structure present in all the images in the dataset. First, the objects in both sides of an image are predicted, starting with the left side; then the model segments the objects in the middle while following similar patterns to the ones in Pascal VOC. This pattern can be observed in Figure 6.2(c).

Further, we quantify the number of object pairs in Pascal VOC images that are predicted in each of the predefined orders. For a pair of objects o_1 and o_2 that are predicted consecutively, we can say they are sorted in a particular order if their difference in the axis of interest is greater than 15% (e.g., a pair of consecutive objects follows a right to left pattern if the second object is to the left of the first by more than $0.15W$ pixels, being W the image width). Figure 6.5 shows the results for object pairs separated by category. For clarity, only pairs of objects that are predicted together at least 20 times are displayed. We observe a substantial difference between pairs of instances from the

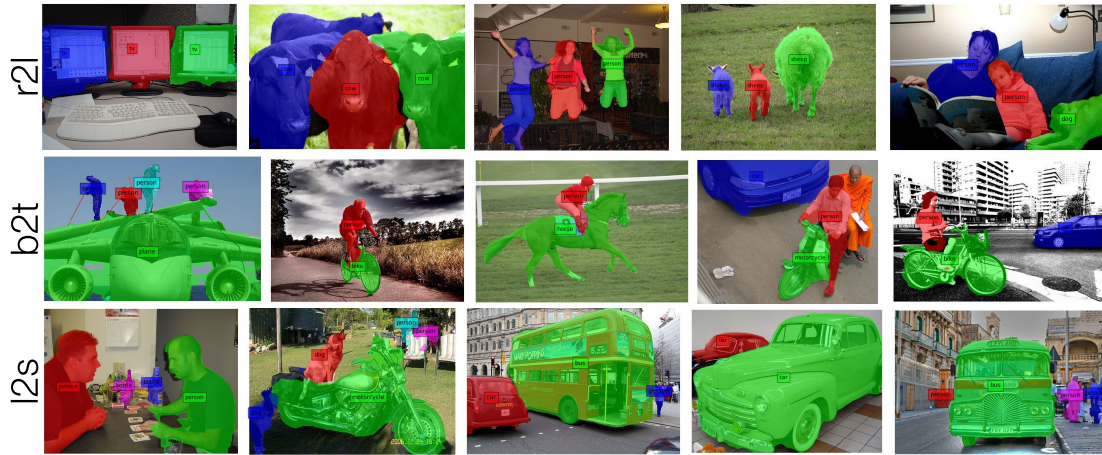


Figure 6.4: Examples of predicted object sequences for images in Pascal VOC 2012 validation set that highly correlate with the different sorting strategies.

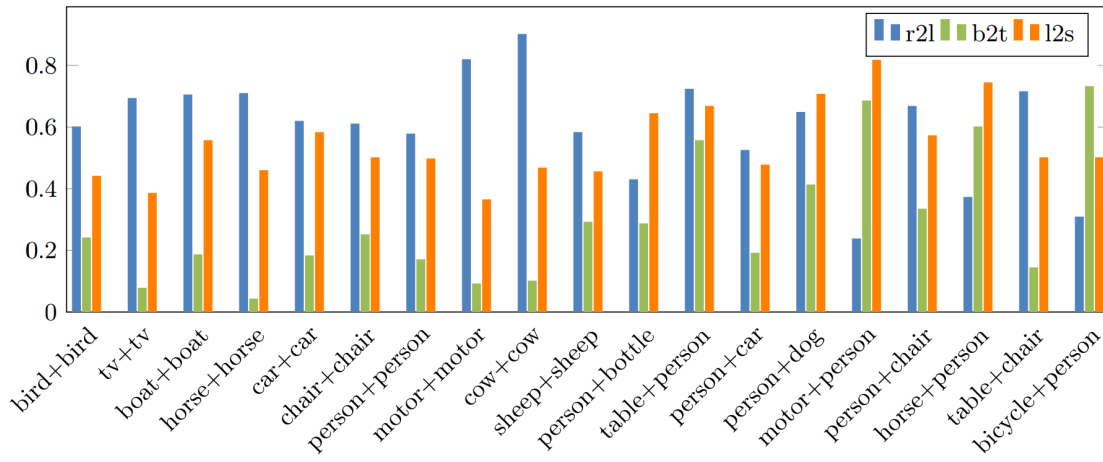


Figure 6.5: Percentage of consecutive object pairs of different categories following a particular sorting pattern.

same category and pairs of objects of different classes. While same-class pairs seem to be consistently predicted following a horizontal pattern (right to left), pairs of objects from different categories are found following other patterns reflecting the relationships between them. For example, the pairs *motorcycle + person*, *bicycle + person* or *horse + person* are often predicted following the vertical axis, from the bottom to the top of the image, which is coherent with the usual spatial distribution of objects of these categories in Pascal VOC images.

We also check whether the order of the predicted object sequences correlates with the features from the encoder. Since these are the inputs to the recurrent layers in the decoder (which do not change across different time steps), the network must learn to encode the information of the object order in these activations. To test whether this is true, we permute the object sequence based on the activations in each of the convolutional layers in the encoder and check the correlation with the original sequence. Table 6.3b shows the Kendall tau correlation values of predicted sequences with these activations, before and

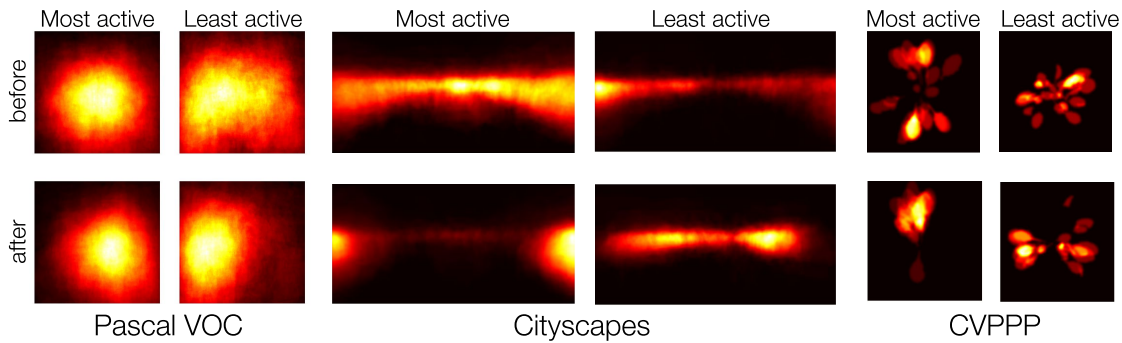


Figure 6.6: Most and least active objects in last (Pascal VOC and Cityscapes) and second to last (CVPPP) block in the encoder before and after training.

after training the model. We observe that correlation increases after training the model for our task. The predicted sequences correlate the most with the activations in the last block in the encoder both for Pascal VOC and Cityscapes. This is a reasonable behavior, since those features are the input to the first ConvLSTM layer in the decoder. In the case of images from the CVPPP dataset, we find that the predicted object sequences correlate with the activations in the second to last convolutional layer in the encoder. We hypothesize that the semantics in the last layer of the encoder, which is pretrained on ImageNet, are not as informative for this task. In Figure 6.6 we display the most and least active object in the most correlated block in the encoder for each dataset. We show figures for features before and after training the model. For Pascal VOC images, we observe a shift of the most active objects from the center of the image to the bottom-right part of the image, while the least active objects are located in the left part of the image. In the case of Cityscapes, the most active objects move from the center to right-most and left-most part of the image after training. Regarding CVPPP, we observe that the network learns a specific route to predict leaves which is consistent across different images, starting in the top-most part of the image.

Conclusion

We have presented a recurrent method for end-to-end semantic instance segmentation, which can naturally handle variable length outputs by construction. Unlike proposal-based methods, which generate an excessive number of predictions and rely on an external post-processing step for filtering them out, our model is able to directly map pixels to the final instance segmentation masks. This allows our model to be optimized for an objective which better matches the conditions of the target task at inference time than those in proposal-based methods. We observed coherent patterns in the order of the predictions that depend on the input image, suggesting that the model makes use of its previous predictions to reason about the next object to be detected. In contrast with other sequential methods that use direct feedback from their output, the choice of a multi-layer recurrent network also has the advantage of being more parallelizable across time steps on modern hardware [5].

Summary

The second part of the thesis has presented methods for set prediction in two different computer vision applications. Chapter 5 presented an overview of existing architectures suitable for multi-label image classification as a set prediction task. Extensive experimentation was conducted on 4 different datasets, which pointed at the superiority of auto-regressive methods for this task. Chapter 6 explores semantic instance segmentation as a set prediction task, proposing an auto-regressive order-agnostic architecture which, in contrast with proposal-based methods, allows to model sets of object masks directly at its output.

In Part III of this thesis, insights gained from Chapter 5 for multi-label image classification will serve as foundations for the ingredient prediction task, which is part of the recipe generation pipeline presented in Chapter 9.

Part III

Image-to-Recipe Prediction

Introduction

7

There are few things so fundamental to the human experience as food. Its consumption is intricately linked to our health, our feelings and our culture [59, 140]. Even migrants starting a new life in a foreign country often hold on to their ethnic food longer than to their native language.

Food culture has been spreading more than ever in the current digital era, with many people sharing pictures of food they are eating across social media [133]. Querying Instagram for #food leads to at least 300M posts; similarly, searching for #foodie results in at least 100M posts, highlighting the unquestionable value that food has in our society. Moreover, eating patterns and cooking culture have been evolving over time. In the past, food was mostly prepared at home, but nowadays we frequently consume food prepared by third-parties (e.g., takeaways, catering and restaurants). Thus, the access to detailed information about prepared food is limited and, as a consequence, it is hard to know precisely what we eat. This creates barriers if one wanted to modify a dish to satisfy constraints such as a sparse pantry or dietary restrictions.

The last few years have witnessed outstanding improvements in visual recognition tasks such as natural image classification [182, 81], object detection [167, 165] and semantic segmentation [125, 90]. However, when comparing to natural image understanding, which has been the focus in the previous chapters of this thesis, food recognition poses additional challenges, since food and its components have high intra-class variability and present heavy deformations that occur during the cooking process. Ingredients are frequently occluded in a cooked dish and come in a variety of colors, shapes and textures. Further, visual ingredient detection requires high level reasoning and prior knowledge (e.g., *cake* will likely contain *sugar* and not *salt*, while *croissant* will presumably include *butter*). Hence, food recognition challenges current computer vision systems to go beyond the merely visible, and to incorporate prior knowledge to enable high-quality structured food preparation descriptions.

The profusion of online recipe collections with user-submitted photos presents the possibility of training machines to automatically understand food preparation by jointly analyzing ingredient lists, cooking instructions and food images. Having such a *multi-modal* understanding of food dishes and their preparation can lead to a sort of personal chef that can intuit a recipe from a picture or a list of ingredients. Far beyond applications solely in the realm of culinary arts, such a tool may also be applied to the plethora of food images shared on social media to achieve insight into the significance of food and its preparation on public health [61] and cultural heritage [135].

This part of the thesis is organized in three chapters. The remainder of this chapter is structured as follows. Section 7.1 reviews previous works on food understanding, while

Section 7.2 introduces the large-scale Recipe1M dataset which will be used to train and evaluate methods presented in this part of the thesis. Basic concepts and state of the art for language modeling, text representations and cross modal applications between language and vision are introduced in sections 7.3, 7.4 and 7.5, respectively. Then, Chapter 8 introduces the image-to-recipe task, and proposes a retrieval-based solution by training a joint neural embedding for images and recipes. In Chapter 9, we pose the image-to-recipe task as conditional long-text generation, in which the recipe (title, ingredients and instructions) is generated directly from the input image.

Food Understanding

The introduction of large scale food datasets, such as Food-101 [19], together with a recently held iFood challenge¹ has enabled significant advancements in visual food recognition, by providing reference benchmarks to train and compare machine learning approaches. As a result, there is currently a vast literature in computer vision dealing with a variety of food related tasks.

Virtually all of the readily available food-related datasets contain only categorized images [136, 19, 97, 216]. For this reason, most works on food understanding focus on image classification [118, 151, 137, 34, 217], by fine-tuning models trained for natural image classification on food recognition datasets annotated with course-level categories [19, 97, 34]. Subsequent works tackled more challenging tasks such as estimating the number of calories given a food image [136] or estimating food quantities [31].

Only recently have a few datasets been released that include both recipes and images. The first of which [205] has 101k images divided equally among 101 categories; the recipes for each are however raw HTML. A later work [29] presented a dataset containing 110,241 images annotated with 353 ingredient labels and 65,284 recipes, each with a brief introduction, ingredient list, and preparation instructions. Of note is that the dataset only contains recipes for Chinese cuisine.

With the release of multi-modal food datasets, several works attempted to solve more challenging tasks such as predicting the list of present ingredients in an image [219, 29, 30] and finding the recipe for a given image [205, 29, 30]. Similarly to [29, 30], Chapter 8 presents a joint neural embedding that allows recipe retrieval from food images. While [29, 30] first predict ingredients and cooking attributes to obtain an intermediate representation used for recipe retrieval, our method is able to directly retrieve both ingredients and instructions using an image embedding. Further, our formulation also allows image retrieval given a recipe.

It is also worth mentioning that food related tasks have also been considered in the natural language processing literature, where recipe generation has been studied in the context of generating procedural text from either flow graphs [74, 144, 143] or ingredients' checklists [98]. In Chapter 9, we propose a method to generate recipes (title, ingredients and instructions) directly from food images.

Despite not being the main focus of this part of the thesis, images and cooking recipes have been utilized beyond the tasks of ingredient prediction and recipe retrieval. Min

¹<https://www.kaggle.com/c/ifood2018>

Partition	# Recipes	# Images
Training	720,639	619,508
Validation	155,036	133,860
Test	154,045	134,338
Total	1,029,720	887,706

Table 7.1: Recipe1M dataset. Number of samples in training, validation and test sets.

et al. [140] provided a detailed cross-region analysis of food recipes, considering images, attributes (e.g., style and course) and recipe ingredients. Recently, Bar El et al. [52] tackle image generation from long structured text in the context of cooking recipes and food images.

Recipe1M Dataset

Recipe1M is the largest dataset of structured recipe data, including over 1M recipes and 800k images. In comparison to the current largest dataset in this domain, Recipe1M includes twice as many recipes as [103] and eight times as many images as [29]. Table 7.1 includes the number of samples included in each of the predefined dataset partitions.

As the ability to learn effective representations is largely a function of the quantity and quality of the available data, this part of the thesis presents methods that are optimized and evaluated on Recipe1M data.

The contents of the Recipe1M dataset may logically be grouped into two layers. The first contains basic information including title, a list of ingredients, and a sequence of instructions for preparing the dish; all of these data are provided as free text. The second layer builds upon the first and includes any images with which the recipe is associated—these are provided as RGB in JPEG format. Additionally, a subset of recipes are annotated with course labels (e.g., appetizer, side dish, dessert), the prevalence of which are summarized in Figure 7.1.

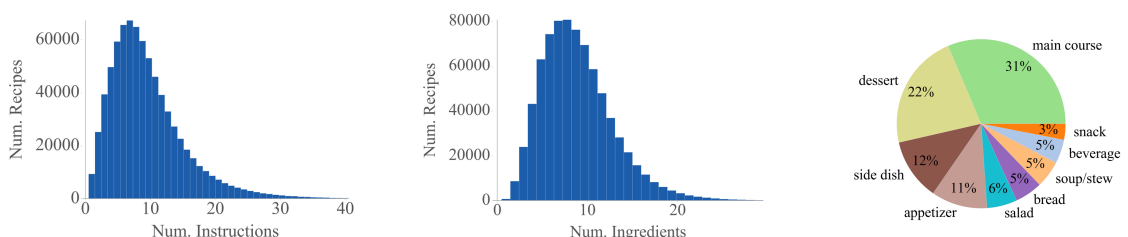


Figure 7.1: Dataset statistics. Prevalence of course categories and number of instructions and ingredients per recipe.

The average recipe in the dataset consists of 9 ingredients which are transformed over the course of 10 instructions. Approximately half of the recipes have images which, due to the nature of the data sources, depict the fully prepared dish. Recipe1M includes approximately 0.4% duplicate recipes and 2% duplicate images (different recipes may share same image). Excluding those 0.4% recipes, 20% of recipes have non-unique titles

but symmetrically differ by a median of 16 ingredients. 0.2% of recipes share the same ingredients but are relatively simple (e.g., spaghetti, granola), having a median of six ingredients. Regarding our experiments, we carefully removed any exact duplicates or recipes sharing the same image in order to avoid overlapping between training and test subsets. As detailed in Table 7.1, around 70% of the data is labeled as training, and the remainder is split equally between the validation and test sets.

In Figure 7.1, one can easily observe that the distributions of data are heavy tailed. For instance, of the 16k unique ingredients that have been identified, only 4,000 account for 95% of occurrences. At the low end of instruction count—particularly those with one step—one will find the dreaded *Combine all ingredients*. At the other end are lengthy recipes and ingredient lists associated with recipes that include sub-recipes. A similar issue of outliers exists also for images: as several of the included recipe collections curate user-submitted images, popular recipes like chocolate chip cookies have orders of magnitude more images than the average. Notably, 25% of images are associated with 1% of recipes while half of all images belong to 10% of recipes; the size of the second layer in number of unique recipes is 333k.

For all the experiments presented in chapters 8 and 9, we use only the recipes containing images, and remove recipes with less than 2 ingredients or 2 instructions, resulting in 252 547 training, 54 255 validation and 54 506 test samples.

Language Modeling

Language Modeling is the task of assigning a probability to sentences in a language, where the likelihood of a given word is estimated based on the words that preceded it [65]. In the field of natural language processing, language models have played a key role in many tasks such as speech recognition, machine translation, text summarization or conversational systems.

Formally, a language model is a probability distribution over a sequence of words:

$$p(w_1, \dots, w_m) \tag{7.1}$$

which is modeled as a conditional probability of words given their context (e.g., its preceding words):

$$p(w_1, \dots, w_m) = \prod_{i=1}^m p(w_i | w_1, \dots, w_{i-1}) \tag{7.2}$$

In practice, neural language models estimate the probability distribution in the logarithmic space, which transforms $\prod_{i=1}^m p(w_i | w_1, \dots, w_{i-1})$ into $\sum_{i=1}^m \log p(w_i | w_1, \dots, w_{i-1})$. Therefore, the goal is to predict \hat{w}_i by maximizing the following objective:

$$\arg \max_{\theta} \sum_{i=1}^m \log p(\hat{w}_i = w_i | w_1, \dots, w_{i-1}, \theta) \tag{7.3}$$

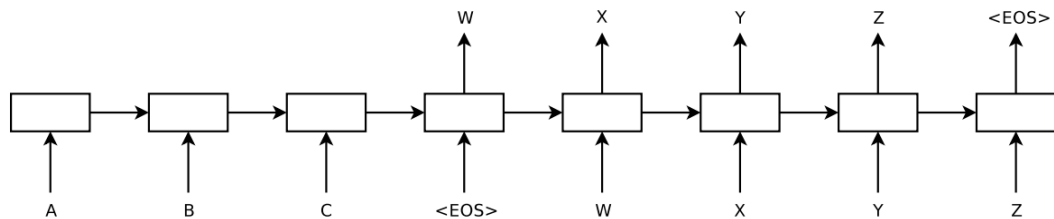


Figure 7.2: Sequence-to-sequence model. Figure from [190].

where θ are the model parameters.

One common metric to measure the quality of a language model is perplexity (ppl), which is defined as:

$$ppl = 2^{\frac{1}{M}H(p)} \quad (7.4)$$

where $H(p)$ is the cross-entropy loss for the sentence: $H(p) = -\sum_{i=1}^m \log p(w_i)$. In plain words, perplexity measures the uncertainty of the language model when predicting a word given the previous words (lower perplexity indicates a better language model).

While language models are often trained to generate text conditioned only on preceding words, one can easily accommodate the objective function to consider an external condition, represented by \mathbf{c} :

$$\arg \max_{\theta} = \sum_{i=1}^m \log p(\hat{w}_i = w_i | w_1, \dots, w_{i-1}, \mathbf{c}, \theta) \quad (7.5)$$

One example of conditional text generation is the machine translation problem, where a sentence in a source language (e.g., English) needs to be translated into a target language (e.g., French). In order to optimize the objective function above to predict a sentence in the target language, one must first encode the sentence in the source language into a representation \mathbf{c} . Models trained for this task are commonly referred to as sequence-to-sequence models [190], which first encode the source sentence into a fix-length representation used to subsequently decode the target sequence. Figure 7.2 shows an overview of the sequence-to-sequence model. More recently, sequence-to-sequence models have been applied to more open-ended generation tasks, such as poetry [206] and story generation [101, 56].

Neural language models are often based on recurrent neural network architectures, which are auto-regressive models by definition (i.e. the output at time-step i depends on its previous outputs $i = \{1, \dots, i-1\}$). However, several works have recently proposed causal convolutions [62] and attention-based models [196] for language modeling, achieving comparable performance and faster training times than their recurrent-based counterparts.

In the spirit of obtaining more realistic text samples, several works have attempted to use generative adversarial networks [41, 57, 221] for text generation.

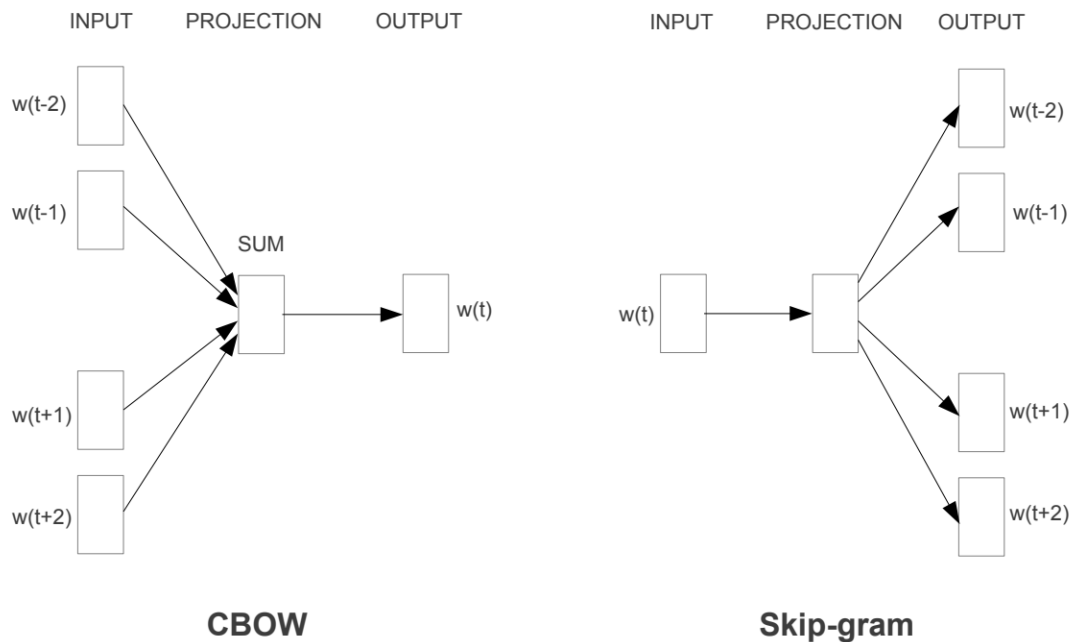


Figure 7.3: Word2vec architectures. Figure taken from [139].

Text Representations

In the field of natural language processing, the most basic representation for text is the one-hot word encoding. Given a vocabulary of words of fixed size $\mathcal{D} = \{d_i\}_{i=0}^N$, we represent the j^{th} word with a vector \mathbf{L} of size N , where $\mathbf{L}_j = 1$ and $\mathbf{L}_{i \neq j} = 0$.

Although the one-hot encoding is commonly used to represent text as an output (where the vocabulary of words is modeled as a categorical distribution), it has some limitations, such as its large dimensionality, which becomes intractable for large-scale vocabularies of millions of words. It is also highly sparse (most elements are 0) and arbitrary (it does not encode word relationships). To overcome these limitations, learning compact word embeddings has been a dominant research topic in natural language processing in the last years [139, 18, 156].

Word embedding models represent words in a continuous vector space in which semantically similar words are mapped to points that are nearby in the embedding space. Word2vec [139] is one of the most popular word embedding models, which is based on the intuition that words appearing in similar contexts within a corpus are semantically similar. Word2vec has two variants, namely the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. While CBOW is trained to predict a target word (e.g., *flying*) from context words (*the bird is*), the skip-gram model is trained in the opposite direction (i.e. to predict the context words given the target). Both CBOW and Skip-Gram are neural language models, which learn to predict a probability distribution over possible words given some context. Figure 7.3 shows the architectures for both models.

Once trained, the learned projections for input words are used as representations to encode them. These representations capture general semantic information about words

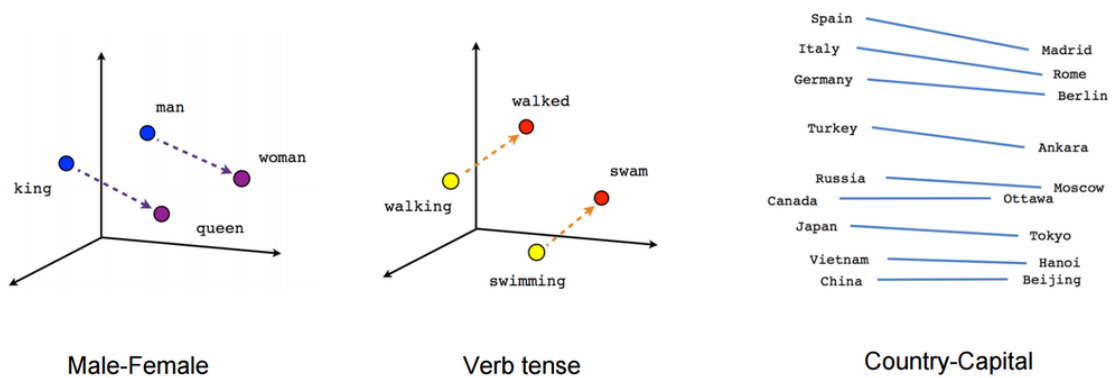


Figure 7.4: Embedding visualization. Figure taken from: <https://www.tensorflow.org/tutorials/representation/word2vec>

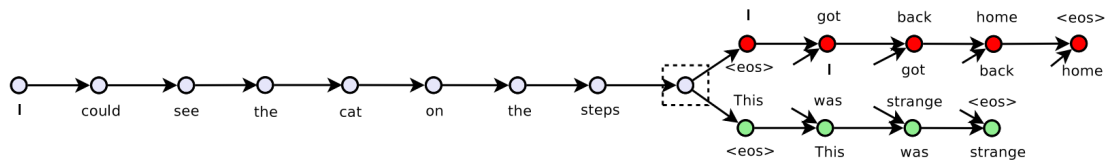


Figure 7.5: Skip thoughts model. Figure from [100].

and their relationships, which has been learned in an unsupervised manner. Figure 7.4 shows examples of visualizations of learned vectors, which exhibit certain properties (e.g., the distance and the direction of the difference vector between female-male word pairs such as *man-woman* and *king-queen* is similar), that explain the semantic-aware nature of these vectors for many tasks e.g., machine translation [190], sentiment analysis [187], or retrieval [10].

One step beyond compact and semantic representations for text (where the encoded unit is a single word), is the skip-thoughts model [100], which extended the skip-gram architecture from [139] to learn sentence embeddings. The model is an LSTM-based encoder-decoder architecture. The LSTM encoder “reads” the input sentence and extracts a representation of fixed size (the hidden state at the last time-step). This hidden state is fed to an LSTM decoder, which predicts the context sentence(s) one word at a time. Figure 7.5 depicts the skip-thoughts model, which is trained on triplets of consecutive sentences in the corpus. Given the middle sentence as an input, the model is optimized to predict the first and the last sentence. Once trained, the representation obtained from the encoder is used to encode sentences, which has been proven useful in a variety of tasks such as sentence classification, image-sentence ranking or paraphrase detection [100].

Language and Vision

Language and vision are two fundamental and complementary modalities through which humans learn and communicate, yet their respective research communities have advanced separately for a long time. Although knowledge has been occasionally transferred from one community to the other (e.g., the bag-of-words model, originally designed to encode text documents, has been widely adopted in computer vision), the intersection between

the two has significantly strengthened in the past decade, thanks to the change in paradigm from hand-crafted to learned representations, and the release of multi-modal datasets [116, 3]. Applications such as visual question answering [3], image captioning [201, 215, 128], visual dialog [45], visual grounding [49], or text-conditioned image generation [131, 224, 84] have emerged and already witnessed remarkable progress in the recent years.

This Section is dedicated to review existing approaches to connect images and language using neural networks. First, Section 7.5.1 reviews joint embedding architectures and loss functions, which will be exploited for recipe retrieval in Chapter 8. Second, Section 7.5.2 reviews existing approaches for image conditioned text generation which will be used in Chapter 9 to generate recipes from images.

Joint Embeddings

Given two sets of data from different modalities x_1 and x_2 , the goal is to obtain a latent representation for each modality $f_1(x_1)$, $f_2(x_2)$ that is suitable for the retrieval of one given the other.

A popular approach for mapping paired sets of vectors into a latent space is the Canonical Correlation Analysis (CCA) [66], which finds a linear projection of the input data by maximizing their correlation:

$$\mathbf{corr}(f_1(x_1), f_2(x_2)) = \frac{\mathbf{cov}(f_1(x_1), f_2(x_2))}{\sqrt{\mathbf{var}(f_1(x_1))\mathbf{var}(f_2(x_2))}} \quad (7.6)$$

Several works have trained deep neural networks with the CCA objective function [2, 27], in order to find non-linear latent spaces to embed input data. However, since correlation is a population objective, gradients must be computed on the full training set, thus prohibiting very deep architectures or big training sets. While recent works have proposed surrogate objectives to train deep CCA models [27], most works currently train joint neural embeddings using paired annotated data.

Joint neural embeddings are optimized with cost functions that minimize the distance between pairs of samples:

$$\min_{f_1, f_2} D(f_1(x_1^{(i)}), f_2(x_2^{(i)})) \quad (7.7)$$

where D is the distance function (e.g., cosine or euclidean). Typically, joint neural embeddings are not only optimized to minimize the distance between positive pairs, but also to maximize the distance between negative pairs.

Popular neural architectures for joint neural embeddings are siamese and triplet networks. Siamese networks [21] are architectures composed of two identical paths that share both their structure and parameters. These networks are trained on paired data $x = \{x_i, x_j\}$, annotated with $Y_{i,j} \in \{0, 1\}$ representing the binary label that indicates whether the

input pair is positive or negative. For each data pair, the contrastive loss is defined as:

$$L_{\text{hinge}}((x_i, x_j), Y_{i,j}) = \begin{cases} D(x_i, x_j), & \text{if } Y_{i,j} = 1 \\ \max(0, \alpha - D(x_i, x_j)), & \text{if } Y_{i,j} = 0 \end{cases} \quad (7.8)$$

where $D(x_i, x_j) = \|x_i - x_j\|_2$ and α is the margin. Instead of the Euclidean distance, several works choose to use the cosine similarity as an alternative metric:

$$L_{\text{cos}}((x_i, x_j), Y_{i,j}) = \begin{cases} 1 - \cos(x_i, x_j), & \text{if } Y_{i,j} = 1 \\ \max(0, \cos(x_i, x_j) - \alpha), & \text{if } Y_{i,j} = 0 \end{cases} \quad (7.9)$$

where $\cos(\cdot)$ is the normalized cosine similarity.

Using the contrastive loss, given a reference data point x_i , a positive example (x_j subject to $Y_{i,j} = 1$) will be pushed closer to x_i , while a negative example (x_j subject to $Y_{i,j} = 0$) will be pushed further apart up to a certain margin α (i.e. a negative data pair does not contribute to the loss if $D(x_i, x_j) > \alpha$).

A generalization of the contrastive loss is the triplet loss, which minimizes relative similarities among triplets of data. Each triplet is composed of a reference example x_r , a positive example x_p and a negative example x_n . In this case, the loss function can be defined as:

$$L_{\text{triplet}}(x_a, x_n, x_p) = \max(0, D(x_a, x_p) - D(x_a, x_n) + \alpha) \quad (7.10)$$

Figure 7.6 illustrates the optimized embedding space for both contrastive and triplet losses.

While originally introduced to optimize siamese and triplet networks on data pairs and triplets from the same modality (e.g., images), these loss functions can be used to optimize joint embeddings in cases where the two network branches are not identical. Several works have trained joint embeddings for images and text for zero-shot image classification [186, 60], sentiment analysis [22] and image captioning [24, 96, 95, 55]. Related to these works, Chapter 8 of this thesis presents a joint embedding that embeds food images and cooking recipes into a common latent space suitable for retrieval. The retrieval performance of the obtained joint embeddings is compared to that of a CCA-based approach, showing their superiority.

Image-conditioned Text Generation

As previously introduced in Section 7.3, conditional text generation with auto-regressive models has been widely studied in the literature for neural machine translation [190, 62, 196, 56]. In these cases, the sentence in the source language is encoded in a context vector \mathbf{c} to optimize the objective function in Equation 7.5. The context vector \mathbf{c} can be obtained using different strategies, e.g., it can be represented by the hidden state of the encoder LSTM [190] or be obtained using an attention layer [196].

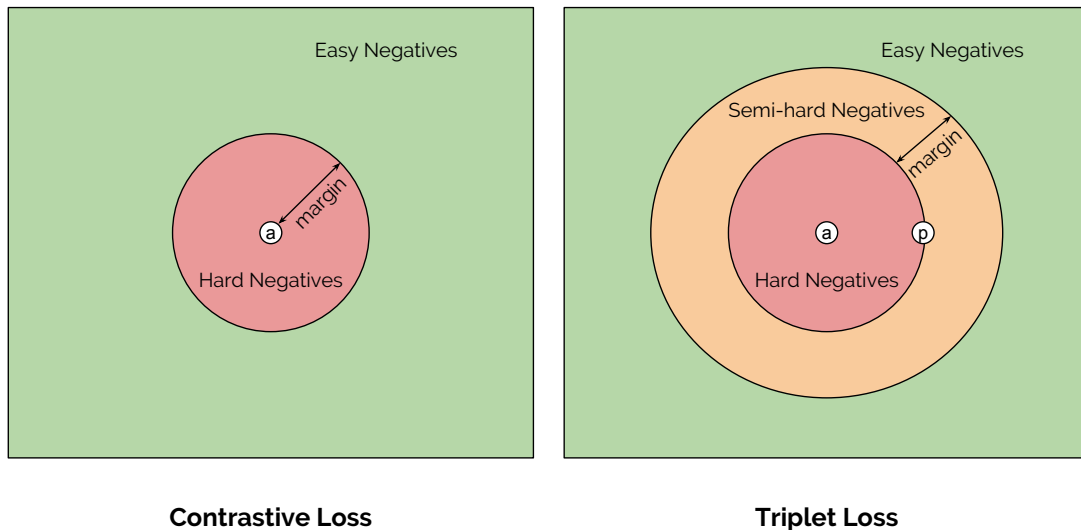


Figure 7.6: Ranking Losses. In the contrastive loss (left), the margin α determines whether a data point is a hard negative relative to its distance to the anchor x_i . Easy negatives satisfy the condition: $D(x_i, x_j) > \alpha$, for which the loss becomes 0. In the triplet loss (right), the negative point is classified relative to the position of the anchor and positive points. The triplet loss becomes 0 for easy negatives, which satisfy $D(x_a, x_p) + \alpha > D(x_a, x_n)$.

Inspired by the success of sequence-to-sequence models for machine translation, several works attempted to generate text with image-based conditionings [201, 215, 128, 95, 101, 41, 179] by extracting the context vector \mathbf{c} using convolutional neural networks [102, 182, 81]. While early attempts used the activations in fully connected layers to encode the image into a fix-length context vector [201, 95], more recent approaches have highlighted the potential of using attention models on top of local convolutional features [215, 128], which give the model the capability of relating image locations with words in the output sequence.

Altogether, auto-regressive models have exhibited promising performance in image captioning [201, 215, 128, 95, 179, 41], where the goal is to provide a short description of the image contents, opening the doors to less constrained problems such as generating descriptive paragraphs [101] or visual storytelling [86]. In this context, Chapter 9 of this thesis presents a novel method to generate cooking recipes (title, ingredients, and instructions) solely from the input food image.

Recipe Retrieval

8

Previously, Chapters 2 and 3 of this thesis have explored image retrieval pipelines based on both off-the-shelf and fine-tuned convolutional features. However, as discussed in the previous chapter (Section 7.5.1) subsequent works in the literature of instance [70], cross-view [110] and text [95] retrieval have highlighted the effectiveness of siamese and triplet losses to train joint embeddings for retrieval. Following insights from these works, yet focusing in the particularities of food images and cooking recipes, in this chapter we present the image-to-recipe (im2recipe) retrieval task, which leverages the full Recipe1M dataset—images and text—to solve the practical and socially relevant problem of demystifying the creation of a dish that can be seen but not necessarily described. To this end, we have developed a deep neural model which jointly learns to embed images and recipes in a common space which is semantically regularized by the addition of a high-level classification task. The performance of the resulting embeddings is thoroughly evaluated against baselines and humans, showing remarkable improvement over the former while faring comparably to the latter. The contributions of this chapter can be summarized as follows:

- We propose a joint neural embedding for cooking recipes and food images, which allows the subsequent retrieval of one given the other. We conduct thorough ablation studies to verify the suitability of our design choices and training procedures.
- We evaluate the learned embeddings on the novel of image-to-recipe retrieval task, achieving state of the art against competitive baselines.
- We thoroughly analyze the learned embeddings, discovering aligned semantic concepts that emerge across modalities after training the model for the im2recipe task.

The remainder of this chapter is structured as follows. Section 8.1 presents the components of our proposed method, including the representations used to encode input data (namely ingredients, instructions and images), the joint neural embedding architecture, and the objective function and training details. Section 8.2 presents the experiments, including ablation studies, comparison with baselines and human performance, and a qualitative embedding analysis. Section 8.3 draws the conclusions.

Methodology

In this section we introduce our neural joint embedding model. Here we utilize the paired (recipe and image) data in order to learn a common embedding space as sketched in Figure 8.1. Next, we discuss recipe and image representations and then we introduce our neural joint embedding model that builds upon recipe and image representations.

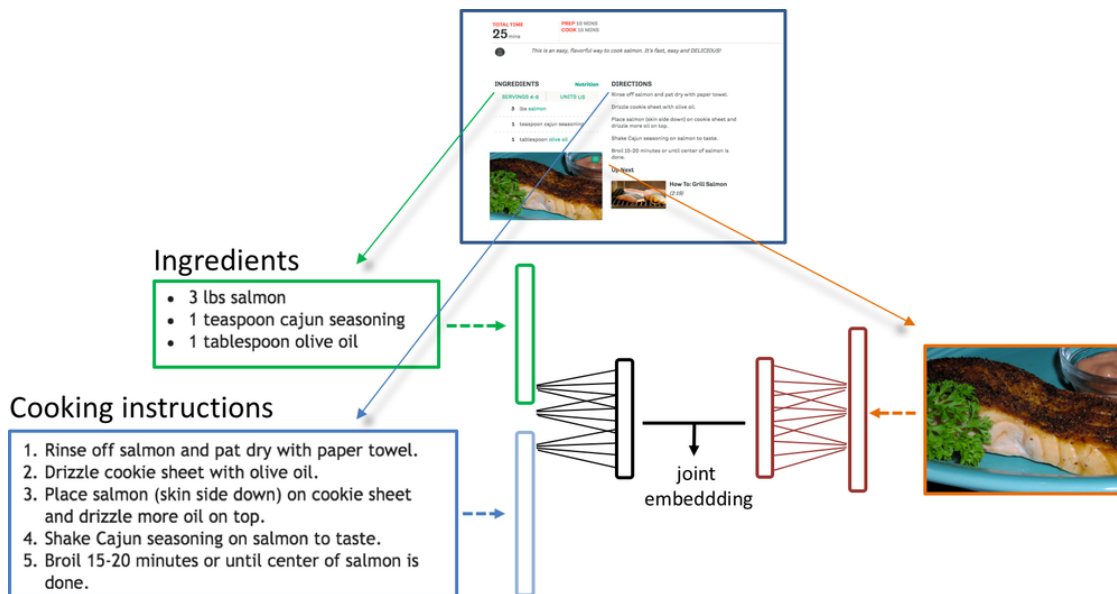


Figure 8.1: Learning cross-modal embeddings from recipe-image pairs collected from online resources. These enable us to achieve in-depth understanding of food from its ingredients to its preparation.

Representation of recipes

There are two major components of a recipe: its ingredients and its cooking instructions. We develop a suitable representation for each of these two components.

Ingredients. Each recipe contains a set of ingredients, where each ingredient is represented with a short sentence (see Figure 8.1). For each ingredient we learn an ingredient level word2vec [139] representation (see Section 7.4 for a review of word2vec). In order to do so, the actual ingredient names are extracted from each ingredient text. For instance in “2 tbsp of olive oil” the *olive_oil* is extracted as the ingredient name and treated as a single word for word2vec computation. The initial ingredient name extraction task is solved by a bi-directional LSTM that performs logistic regression on each word in the ingredient text. Training is performed on a small subset of our training set for which we have the annotation for actual ingredient names. Ingredient name extraction module works with 99.5% accuracy tested on a held-out set. Given a sample in the Recipe1M dataset, the input to the joint embedding model on the ingredient side is composed with the word2vec representation of each ingredient in the set.

Cooking Instructions. Each recipe also has a list of cooking instructions. As the instructions are quite lengthy (averaging 208 words) a single LSTM is not well suited to their representation as gradients are diminished over the many time steps. Instead, we propose to encode a recipe as a sequence of sequences, where the first step is to encode each instruction in the recipe into fixed-size representation. Our cooking instruction representation, referred as *skip-instructions*, is the product of a sequence-to-sequence model [190]. Specifically, we build upon the technique of skip-thoughts [100] (explained in Section 7.4) which encodes a sentence and uses that encoding as context when decoding/predicting the previous and next sentences. We train the skip-thoughts model on the training set of Recipe1M and use the output of the encoder as the representation for

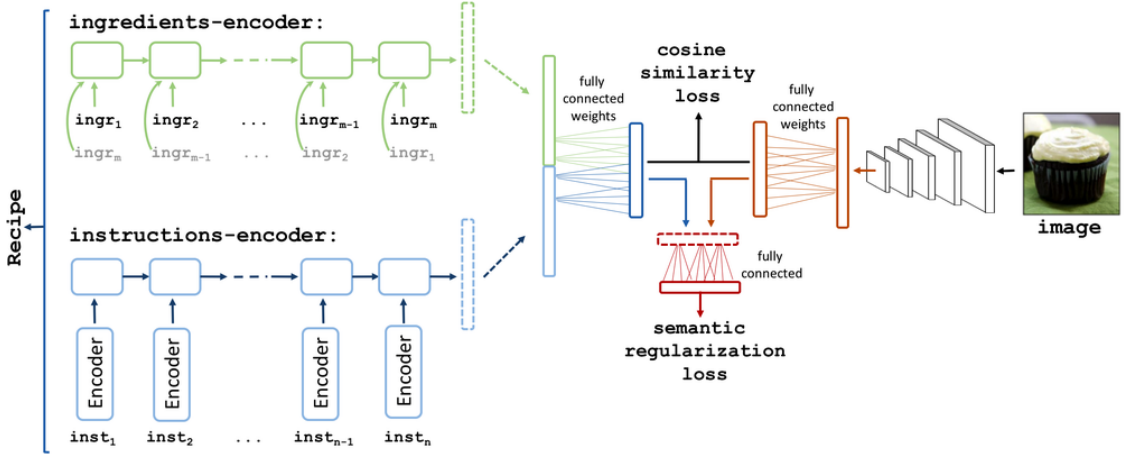


Figure 8.2: Joint neural embedding model with semantic regularization. Our model learns a joint embedding space for food images and cooking recipes.

each instruction in the recipe. These representations are used as inputs to our embedding model (see instructions-encoder in Figure 8.2).

Representation of food images

For the image representation we adopt two major state-of-the-art deep convolutional networks, namely VGG-16 [182] and ResNet-50 [81] models. We integrate these models in our joint embedding by removing the last softmax classification layer and connecting the rest to our joint embedding model as shown in the right side of Figure 8.2. We train VGG-16 and ResNet-50 versions of our model and compare them in Section 8.2.1.

Joint Neural Embedding

Building upon the previously described recipe and image representations, we now introduce our joint embedding method. The recipe model, displayed in Figure 8.2, includes two encoders: one for ingredients and one for instructions, the combination of which are designed to learn a recipe level representation.

The goal is to learn transformations to make the embeddings for a given recipe-image pair “close.” Formally, assume that we are given a set of the recipe-image pairs, (R_k, v_k) in which R_k is the k^{th} recipe and v_k is the associated image. Further, let $R_k = (\{s_k^t\}_{t=1}^{n_k}, \{g_k^t\}_{t=1}^{m_k}, v_k)$, where $\{s_k^t\}_{t=1}^{n_k}$ is the sequence of n_k cooking instructions, $\{g_k^t\}_{t=1}^{m_k}$ is the sequence of m_k ingredient tokens. The objective is to maximize the cosine similarity between positive recipe-image pairs, and minimize it between all non-matching recipe-image pairs, up to a specified margin.

The ingredients encoder is implemented using a bi-directional LSTM: at each time step it takes two ingredient-word2vec representations of g_k^t and g_k^{m-t+1} , and eventually it produces the fixed-length representation h_k^g for ingredients. The instructions encoder is implemented through a regular LSTM. At each time step it receives an instruction representation from the skip-instructions encoder, and finally it produces the fixed-length representation h_k^s . h_k^g and h_k^s are concatenated in order to obtain the recipe representation h_k^R . Then the recipe and image representations are mapped into the joint embedding

space as: $\phi^R = W^R h_k^R + b^R$ and $\phi^v = W^v v_k + b^v$, respectively. W^R and W^v are embedding matrices which are also learned. Finally the complete model is trained end-to-end with positive and negative recipe-image pairs (ϕ^R, ϕ^v) using the cosine similarity loss with margin (defined in Section 7.5.1).

	im2recipe				recipe2im			
	medR	R@1	R@5	R@10	medR	R@1	R@5	R@10
random ranking	500	0.001	0.005	0.01	500	0.001	0.005	0.01
CCA w/ skip-thoughts + w2v (GoogleNews) + image features	25.2	0.11	0.26	0.35	37.0	0.07	0.20	0.29
CCA w/ skip-instructions + ingredient w2v + image features	15.7	0.14	0.32	0.43	24.8	0.09	0.24	0.35
joint emb. only	7.2	0.20	0.45	0.58	6.9	0.20	0.46	0.58
joint emb. + sem.	5.2	0.24	0.51	0.65	5.1	0.25	0.52	0.65

Table 8.1: im2recipe retrieval comparisons. Median ranks and recall rate at top K are reported for baselines and our method. Note that the joint neural embedding models consistently outperform all the baseline methods.

	Joint emb. methods	im2recipe			recipe2im		
		medR-1K	medR-5K	medR-10K	medR-1K	medR-5K	medR-10K
VGG-16	fixed vision	15.3	71.8	143.6	16.4	76.8	152.8
	finetuning (ft)	12.1	56.1	111.4	10.5	51.0	101.4
	ft + semantic reg.	8.2	36.4	72.4	7.3	33.4	64.9
ResNet-50	fixed vision	7.9	35.7	71.2	9.3	41.9	83.1
	finetuning (ft)	7.2	31.5	62.8	6.9	29.8	58.8
	ft + semantic reg.	5.2	21.2	41.9	5.1	20.2	39.2

Table 8.2: Ablation studies. Effect of the different model components to the median rank (the lower is better).

Semantic Regularization

We incorporate additional regularization on our embedding through solving the same high-level classification problem in multiple modalities with shared high-level weights. We refer to this method as semantic regularization. The key idea is that if high-level discriminative weights are shared, then both of the modalities (recipe and image embeddings) should utilize these weights in a similar way which brings another level of alignment based on discrimination. We optimize this objective together with our joint embedding loss. Essentially the model also learns to classify any image or recipe embedding into one of the food-related semantic categories.

We start by assigning Food-101 categories to those recipes that contain them in their title. However, after this procedure we are only able to annotate 13% of our dataset, which we argue is not enough labeled data for a good regularization. Hence, we compose a larger set of semantic categories purely extracted from recipe titles. We first obtain the top 2,000 most frequent bigrams in recipe titles from our training set. We manually remove those that contain unwanted characters (e.g., n' , $!$ or $?$) and those that do not have discriminative food properties (e.g., *best pizza*, *super easy* or *5 minutes*). We then assign each of the remaining bigrams as the semantic category to all recipes that include it in their title. By using bigrams and Food-101 categories together we obtain a total of 1,047 categories, which cover 50% of the dataset. *chicken salad*, *grilled vegetable*,

chocolate cake and *fried fish* are some examples among the categories we collect using this procedure. All those recipes without a semantic category are assigned to an additional *background* class. Although there is some overlap in the generated categories, 73% of the recipes in our dataset (excluding those in the *background* class) belong to a single category (i.e., only one of the generated classes appears in their title). For recipes where two or more categories appear in the title, the category with highest frequency rate in the dataset is chosen.

To incorporate semantic regularization to the joint embedding we use a single fully connected layer. Given the embeddings ϕ^v and ϕ^r , class probabilities are obtained with $p_r = W^c \phi^r$ and $p_v = W^c \phi^v$ followed by a softmax activation. W^c is the matrix of learned weights, which are shared between image and recipe embeddings to promote semantic alignment between them. Formally, we express the semantic regularization loss as $L_{reg}(\phi^r, \phi^v, c_r, c_v)$ where c_r, c_v are the semantic category labels for recipe and image, respectively. Note that c_r and c_v are the same if (ϕ^r, ϕ^v) is a positive pair. Then we can write the final objective as:

$$L(\phi^r, \phi^v, c_r, c_v, y) = L_{cos}((\phi^r, \phi^v), y) + \lambda L_{reg}(\phi^r, \phi^v, c_r, c_v)$$

Training Details

We follow a two-stage optimization procedure while learning the model. If we update both the recipe encoding and image network at the same time, optimization becomes oscillatory and even divergent. Previous work on cross-modality training [26] suggests training models for different modalities separately and fine tuning them jointly afterwards to allow alignment. Following this insight, we adopt a similar procedure when training our model. We first fix the weights of the image network, which are found from pre-training on the ImageNet object classification task, and learn the recipe encodings. This way the recipe network learns to align itself to the image representations and also learns semantic regularization parameters (W^c). Then we freeze the recipe encoding and semantic regularization weights, and learn the image network. This two-stage process is crucial for successful optimization of the objective function. After this initial alignment stage, we release all the weights to be learned. However, the results do not change much in this final, joint optimization.

All the neural models are implemented using the Torch7 framework¹. The margin α for the cosine loss Equation 7.9 is set as 0.1 in joint neural embedding models. The regularization hyperparameter from Equation 8.1.4 is set as $\lambda = 0.02$ in all our experiments. While optimizing the cosine loss we pick a positive recipe-image pairs with 20% probability and a random negative recipe-image pair with 80% probability from the training set. The models are trained on 4 NVIDIA Titan X with 12GB of memory for three days.

Experiments

We begin with the evaluation of our learned embeddings for the im2recipe retrieval task. We then study the effect of each component of our model and compare our final system

¹<http://torch.ch/>

against human performance. We also analyze the properties of our learned embeddings through unit visualizations and vector arithmetics in the embedding space.






im2recipe retrieval

We evaluate all the recipe representations for im2recipe retrieval. Given a food image, the task is to retrieve its recipe from a collection of test recipes. We also perform recipe2im retrieval using the same setting. All results are reported for the test set.





















Comparison with the baselines. We compare our joint embedding with a Canonical Correlation Analysis (CCA) baseline (introduced in Section 7.5.1). CCA embeddings are learned using recipe-image pairs from the training data. In each recipe, the ingredients are represented with the mean word2vec across all its ingredients in the manner of [105]. The cooking instructions are represented with mean skip-thoughts vectors [100] across the cooking instructions. A recipe is then represented as concatenation of these two features. We also evaluate CCA over mean ingredient word2vec and skip-instructions features as another baseline. The image features utilized in the CCA baselines are the ResNet-50 features before the softmax layer. Although they are learned for visual object categorization tasks on ImageNet dataset, these features are widely adopted by the computer vision community, and they have been shown to generalize well to different visual recognition tasks [50].

For evaluation, given a test query image, we use cosine similarity in the common space for ranking the relevant recipes and perform im2recipe retrieval. The recipe2im retrieval setting is evaluated likewise. We adopt the test procedure from image2caption retrieval task [95, 201]. We report results on a subset of randomly selected 1,000 recipe-image pairs from the test set. We evaluate on 10 subsets of randomly selected 1 000 recipe-image pairs from the test set and report the mean results. We report median rank (MedR), and recall rate at top K (R@K) for all the retrieval experiments. To clarify, R@5 in the im2recipe task represents the percentage of all the image queries where the corresponding recipe is retrieved in the top 5, hence higher is better. The quantitative results for im2recipe retrieval are shown in Table 8.1.

Our model greatly outperforms the CCA baselines in all measures. As expected, CCA over ingredient word2vec and skip-instructions perform better than CCA over word2vec trained on GoogleNews [139] and skip-thoughts vectors are learned over a large-scale book corpus [100]. In 65% of all evaluated queries, our method can retrieve the correct recipe given a food image. The semantic regularization notably improves the quality of our embedding for im2recipe task which is quantified with the medR drop from 7.2 to 5.2 in Table 8.1. The results for recipe2im task are also similar to those in the im2recipe retrieval setting. Figure 8.4 compares the ingredients from the original recipes (true recipes) with the retrieved recipes (coupled with their corresponding image) for different image queries. We observe that our embeddings generalize well and allow particular recipe retrieval (e.g., smoothie, lasagna and sushi or hamburger in the figure). However, in some cases our model retrieves recipes missing ingredients due to the lack of fine grained features (e.g., confusion between *shrimp sushi* and *salmon sushi*, missing *mayonnaise* or *carrots* in a hamburger recipe) or due to the ambiguity coming from the task itself, which can be specially challenging when ingredients are not visible in the query image of a cooked dish (e.g., *beef* in a lasagna or *blueberries* in a smoothie). In Figure 8.3 we provide more examples of im2recipe and recipe2m retrieval scenarios, including full

image query	retrieved recipe	
	sour_cream white_sugar cream_of_tartar vanilla_wafers vanilla_extract cream_cheese	Preheat an oven to 350 degrees F (175 degrees C). Wrap the exterior of an 8-inch springform pan with aluminum foil, then spray the interior with cooking spray. Spread the vanilla wafer crumbs over the bottom and sides of the pan, pressing firmly to adhere. Cream together the egg yolks and sugar in a large bowl. Stir in the cream cheese, sour cream, and vanilla extract until smooth. Beat egg whites until foamy in a large glass or metal mixing bowl.
	beer lemonade vodka	Mix a 2 quart lemonade powder mix with less than 2 quarts of water (strong lemonade). Add beer and vodka and mix. Serve with ice and enjoy!
	onion eggs cream brown_sugar garlic chicken_broth	Crumble the ground beef into a large skillet over medium heat. Add onions and garlic; cook and stir until evenly browned. Drain off excess fat. Stir in the red wine, tomatoes, oregano, red pepper flakes and sea salt. Bring to a boil, then reduce heat to low, and simmer for 20 minutes. Preheat the oven to 350 degrees F (175 degrees C).
	raw_shrimp pea_pods butter sliced_mushrooms orange_sections soy_sauce	Cook noodles according to package directions. Drain. Keep warm, set aside. While noodles are cooking, pour mushroom and orange liquids into a measuring cup add water to equal 1 cup. Blend in cornstarch and set aside. In a large skillet, melt butter and saute mushrooms, shrimp, pea pods, green onions, and garlic powder
	canola_oil poblano_chiles buns ground_chuck provolone_cheese black_pepper	Heat the grill to high. Brush the chiles with oil and season with salt and pepper. Grill until blistered on all sides and soft, about 12 minutes. Remove the chiles from the grill, place in a bowl, cover with plastic wrap and let sit 15 minutes. Peel, stem and seed the chiles. Slice the chiles and set aside.

(a) im2recipe

recipe query (ingredients + instructions)	top 4 retrieved images
plums cinnamon butter vanilla_extract salt milk	   
kosher_salt grapefruit_juice lime_juice chipotle_powder tequila lime_zest	   
carrots fresh_spinach lasagna_noodles ricotta_cheese zucchini salt	   
onion asparagus stock olive_oil garlic broccoli	   
beef jalapeno tomato guacamole lettuce_leaf avocados	   

(b) recipe2im

Figure 8.3: Retrieval examples.

recipes (ingredients and instructions).

Ablation studies. We also analyze the effect of each component in our model in several optimization stages. The results are reported in Table 8.2. Note that here we also report medR with 1K, 5K and 10K random selections to show how the results scale in larger retrieval problems. As expected, visual features from the ResNet-50 model









Query Image	True ingrs.	Retrieved ingrs.	Retrieved Image
	whole milk half - and - half cr white sugar lemon extract ground cinnamon frozen blueberries vanilla wafers ice cubes	berries strawberry yogurt banana milk white sugar	
	butter garlic cloves all - purpose flour kosher salt milk chicken broth mozzarella cheese parmesan cheese onion	1 box any pasta you ground beef 1 envelope taco seas water 1/2 packages cream c cheese	
	cooked white rice salt shrimp Broccolini mayonnaise nori	sushi rice salmon avocado cream cheese nori	
	mayonnaise onion cider vinegar sugar celery seeds green cabbage carrot salt & freshly groun ground chuck	yellow onion coarse salt ground pepper ground chuck buns eggs ketchup canned beets lettuce leaves	

Figure 8.4: Retrieval examples. From left to right: (1) the query image, (2) its true ingredients, (3) the retrieved ingredients and (4) the image from the retrieved recipe.

show a substantial improvement in retrieval performance when compared to VGG-16 features. Even with “fixed vision” networks (i.e. trained freezing the weights of the image encoder), the joint embedding achieved 7.9 medR using ResNet-50 architecture (see Table 8.2). Further “finetuning” of vision networks slightly improves the results. Although it becomes a lot harder to decrease the medR in small numbers, additional “semantic regularization” improves the medR in both cases.

Comparison with human performance

In order to better assess the quality of our embeddings we also evaluate the performance of humans on the im2recipe task. The experiments are performed through Amazon Mechanical Turk (AMT) service². For quality purposes, we require each AMT worker to have at least 97% approval rate and have performed at least 500 tasks before our experiment. In a single evaluation batch, we first randomly choose 10 recipes and their corresponding images. We then ask an AMT worker to choose the correct recipe, out of the 10 provided recipes, for the given food image. This multiple choice selection task is performed 10 times for each food image in the batch. The accuracy of an evaluation batch is the percentage of image queries correctly assigned to their corresponding recipe.

The evaluations are performed for three levels of difficulty. The batches (of 10 recipes) are randomly chosen from either all the test recipes (easy), recipes sharing the same course (e.g., soup, salad, or beverage; medium), or recipes sharing the name of the dish (e.g., salmon, pizza, or ravioli; hard). As expected—for our model as well as the AMT workers—the accuracies decrease as tasks become more specific. In both coarse and fine-

²<http://mturk.com>

	all recipes	course-specific recipes							dish-specific recipes							
		dessert	salad	bread	drink	soup	mean	pasta	pizza	steak	salmon	smoothie	burger	ravioli	sushi	mean
human	81.6 ± 8.9	52.0	70.0	34.0	58.0	56.0	54.0 ± 13.0	54.0	48.0	58.0	52.0	48.0	46.0	54.0	58.0	52.2 ± 04.6
joint-emb. only	83.6 ± 3.0	76.0	68.0	38.0	24.0	62.0	53.6 ± 21.8	58.0	58.0	58.0	64.0	38.0	58.0	62.0	42.0	54.8 ± 09.4
joint-emb.+sem.	84.8 ± 2.7	74.0	82.0	56.0	30.0	62.0	60.8 ± 20.0	52.0	60.0	62.0	68.0	42.0	68.0	62.0	44.0	57.2 ± 10.1

Table 8.3: Comparison with human performance on im2recipe task. The mean results are highlighted as bold for better visualization. Note that on average our method with semantic regularization performs better than average AMT worker.

	Top 4 images	Top 2 ingredients	Top 2 instructions
unit 352		vanilla_extract, nutmeg, heavy_cream, creme_fraiche, all_purpose_flour, sugar, potatoes, garlic, cloves, chunks	Start with bowl and beaters cold! In a large bowl, whip cream until stiff peaks are ju... Beat in vanilla and sugar until stiff peaks form. Do not overbeat! Put flour and salt in a mixing bowl for use a food p... Add half the butter and mix well, until mixture rese... Add remaining butter chunks and the water and mix un... Remove dough, divide into two equal pieces and dust... Quickly form each piece into a ball, then press down... Wrap and refrigerate for at least an hour.
unit 386		tomatoes, carrots, garlic, cashews, dates, leaf, milk, vinegar, sugar, tomato_paste	Coat pan with cooking oil and pan fry Mahi Mahi fill... To prepare sauce, saute garlic and shallots in pan. Stir in chicken stock and simmer until sauce thickens. Remove from heat and add basil. To Serve, top Mahi Mahi fillets with generous helpin... Garnish with a pretty whole basil leaf or bunch of L...
unit 144		onion, mung_bean, fresh_spinach, mushroom, chile_pepper, olive_oil, soy_sauce, vegetable_oil, coconut_milk, black_pepper	Fry bacon in a Dutch oven until almost done. Add onions and garlic and saute until the onions are... Cover the bacon, onions and garlic with 4 cups water... Add wine, soy sauce, salt, hot sauce and collards. Return to a boil and simmer for 1 hour.
unit 22		butter, pudding, milk, almond_extract, vanilla, water, blend, yellow_cake_mix, oil, baking_powder, powdered_sugar, sugar	Preheat oven to 350F. Beat butter and sugar in large bowl with electric mi... Add eggs, one at a time, beating well after each add... Add cheese and sour cream; mix well. Bake 40 min. Cool completely. Heat oven to 350F. Beat all ingredients except sugar with mixer until b... Pour into greased and floured 12-cup fluted tube pan... Bake 50 to 55 min. or until toothpick inserted near center comes out cl... Cool cake completely.
unit 571		steak, green_pepper, garlic_powder, swiss_cheese, brown_sugar, steak, onion_powder, italian_dressing, roast, tomato_paste, black_pepper, beef_broth	Heat grill to medium heat. Mix all ingredients except steaks; rub onto both sid... Grill 6 to 8 min. Remove from grill. Let stand 5 min. before serving. Sprinkle generously with salt and pepper on both sides. Add the onion and apples. Sauté until the onion slices are lightly caramelized... Cook until the pork is tender, about 15 more minutes... If the apple mixture needs a little thickening, tran... Serve the chops over rice or mashed potatoes with a...

Figure 8.5: Localized unit activations. We find that ingredient detectors emerge in different units in our embeddings, which are aligned across modalities (e.g., unit 352: “cream”, unit 22: “sponge cake” or unit 571: “steak”).

grained tests, our method performs comparably to or better than the AMT workers. As hypothesized, semantic regularization further improves the results (see Table 8.3).

In the “all recipes” condition, 25 random evaluation batches (25×10 individual tasks in total) are selected from the entire test set. Joint embedding with semantic regularization performs the best with 3.2 percentage points improvement over average human accuracy. For the course-specific tests, 5 batches are randomly selected within each given meal course. Although, on average, our joint embedding’s performance is slightly lower than the humans’, with semantic regularization our joint embedding surpasses humans’ performance by 6.8 percentage points. In dish-specific tests, five random batches are selected if they have the dish name (e.g., pizza) in their title. With slightly lower accuracies in general, dish-specific results also show similar behavior. Particularly for the “beverage” and “smoothie” results, human performance is better than our method, possibly because detailed analysis is needed to elicit the homogenized ingredients in drinks. Similar behavior is also observed for the “sushi” results where fine-grained features of the sushi roll’s center are crucial to identify the correct sushi recipe.

Analysis of the learned embedding

To gain further insight into our neural embedding, we perform a series of qualitative analysis experiments. We explore whether any semantic concepts emerge in the neuron activations and whether the embedding space has certain arithmetic properties.



Figure 8.6: Arithmetics using image embeddings (top) and recipe embeddings (bottom). We represent the average vector of a query with the images from its 4 nearest neighbors. In the case of the arithmetic result, we show the nearest neighbor only.

Neuron Visualizations. Through neural activation visualization we investigate if any semantic concepts emerge in the neurons in our embedding vector despite not being explicitly trained for that purpose. We pick the top activating images, ingredient lists, and cooking instructions for a given neuron. Then we use the methodology introduced by Zhou et al. [231] to visualize image regions that contribute the most to the activation of specific units in our learned visual embeddings. We apply the same procedure on the recipe side to also obtain those ingredients and recipe instructions to which certain units react the most. Figure 8.5 shows the results for the same unit in both the image and recipe embedding. We find that certain units display localized semantic alignment between the embeddings of the two modalities.

Semantic Vector Arithmetic. Different works in the literature [139, 163] have used simple arithmetic operations to demonstrate the capabilities of their learned representations. In the context of food recipes, one would expect that $v(\text{“chicken pizza”}) - v(\text{“pizza”}) + v(\text{“salad”}) = v(\text{“chicken salad”})$, where v represents the map

into the embedding space. We investigate whether our learned embeddings have such properties by applying the previous equation template to the averaged vectors of recipes that contain the queried words in their title. We apply this procedure in the image and recipe embedding spaces and show results in Figures 8.6(a) and 8.6(b), respectively. Our findings suggest that the learned embeddings have semantic properties that translate to simple geometric transformations in the learned space.

Conclusion

In this chapter, we presented the im2recipe problem, and neural embedding models with semantic regularization which achieve impressive results for the im2recipe task. More generally, the methods presented here could be gainfully applied to other “recipes” like assembly instructions, tutorials, and industrial processes. Further, we hope that our contributions will support the creation of automated tools for food and recipe understanding and open doors for many less explored aspects of learning such as compositional creativity and predicting visual outcomes of action sequences.

Recipe Generation

9

In the previous chapter, the image-to-recipe problem was formulated as a retrieval task [205, 29, 30], where a recipe is retrieved from a fixed dataset based on the image similarity score in an embedding space. However, the performance of retrieval-based systems highly depends on the dataset size and diversity, as well as on the quality of the learned embedding. Not surprisingly, these systems fail when a matching recipe for the image query does not exist in the static dataset.

An alternative to overcome the dataset constraints of retrieval systems is to formulate the image-to-recipe problem as a conditional generation one. Therefore, in this chapter, we present a system that *generates* a cooking recipe containing a title, ingredients and cooking instructions directly from an image. To the best of our knowledge, our system is the first to *generate* cooking recipes directly from food images. We pose the instruction generation problem as a sequence generation one *conditioned on two modalities* simultaneously, namely an image and its predicted ingredients. We formulate the ingredient prediction problem as a *set prediction*, exploiting their underlying structure. We model ingredient dependencies while not penalizing for prediction order, thus revising the question of *whether order matters* [198]. We extensively evaluate our system on the Recipe1M dataset that contains images, ingredients and cooking instructions, showing satisfactory results. More precisely, in a human evaluation study, we show that our inverse cooking system outperforms previously introduced image-to-recipe retrieval approaches by a large margin. Moreover, using a small set of images, we show that food image-to-ingredient prediction is a hard task for humans and that our approach is able to surpass them.

The contributions of this chapter can be summarized as:

- We present an inverse cooking system, which generates cooking instructions conditioned on an image and its ingredients, exploring different attention strategies to reason about both modalities simultaneously.
- We exhaustively study ingredients as both a *list* and a *set*, and propose a new architecture for ingredient prediction that exploits co-dependencies among ingredients without imposing order.
- By means of a user study we show that ingredient prediction is indeed a difficult task and demonstrate the superiority of our proposed system against image-to-recipe retrieval approaches.

The remainder of this chapter is structured as follows. Section 9.1 describes the components of our method, namely the cooking instructions decoder (Section 9.1.1), the ingredient prediction module (Section 9.1.2), and optimization details (Section 9.1.3).

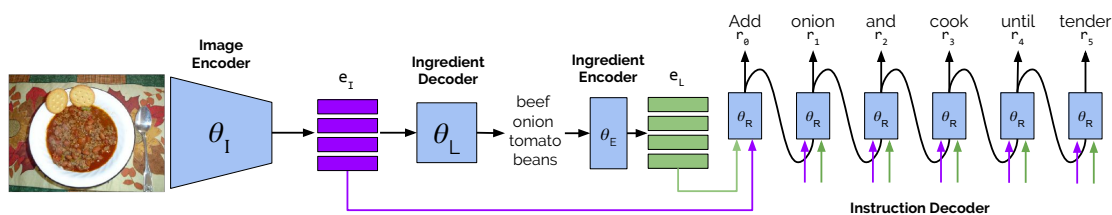


Figure 9.1: Recipe generation model. We extract image features \mathbf{e}_I with the image encoder, parametrized by θ_I . Ingredients are predicted by θ_L , and encoded into ingredient embeddings \mathbf{e}_L with θ_e . The cooking instruction decoder, parametrized by θ_R generates a recipe title and a sequence of cooking steps by attending to image embeddings \mathbf{e}_I , ingredient embeddings \mathbf{e}_L , and previously predicted words (r_0, \dots, r_{t-1}) .

Section 9.2 presents the experiments, including an analysis of each of the components, comparison with retrieval baselines and user studies. Section 9.3 draws the conclusions of this chapter.

Methodology

Generating a recipe (title, ingredients and instructions) from an image is a challenging task, which requires a simultaneous understanding of the ingredients composing the dish as well as the transformations they went through, e.g., slicing, blending or mixing with other ingredients. Instead of obtaining the recipe from an image directly, we argue that a recipe generation pipeline would benefit from an intermediate step predicting the ingredients list. The sequence of instructions would then be generated conditioned on both the image and its corresponding list of ingredients, where the interplay between image and ingredients could provide additional insights on how the latter were processed to produce the resulting dish.

Figure 9.1 illustrates our approach. Our recipe generation system takes a food image as an input and outputs a sequence of cooking instructions, which are generated by means of an instruction decoder that takes as input two embeddings. The first one represents visual features extracted from an image, while the second one encodes the ingredients extracted from the image. We start by introducing our transformer-based instruction decoder in Subsection 9.1.1. This allows us to formally review the transformer, which we then study and modify to predict ingredients in an orderless manner in Subsection 9.1.2. Finally, we review the optimization details in Subsection 9.1.3.

Cooking Instruction Transformer

Given an input image with associated ingredients, we aim to produce a sequence of instructions $R = (r_1, \dots, r_T)$ (where r_t denotes a word in the sequence) by means of an instruction transformer [196]. Note that the title is predicted as the first instruction. This transformer is conditioned jointly on two inputs: the image representation \mathbf{e}_I and the ingredient embedding \mathbf{e}_L . We extract the image representation with a ResNet-50 [81] encoder and obtain the ingredient embedding \mathbf{e}_L by means of a decoder architecture to predict ingredients (see Section 9.1.2), followed by a single embedding layer mapping each ingredient into a fixed-size vector.

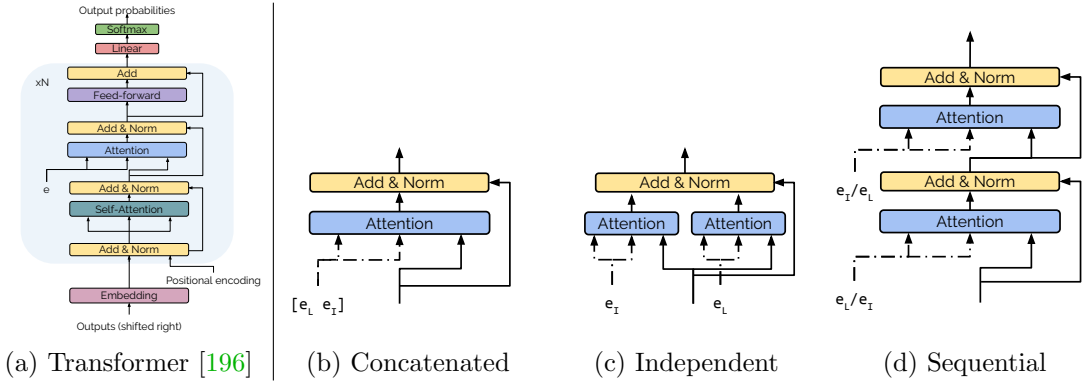


Figure 9.2: Attention strategies for the instruction decoder. In our experiments, we replace the attention module in the transformer (a), with three different attention modules (b-d) for cooking instruction generation using multiple conditions.

The instruction decoder is composed of *transformer blocks*, each of them containing two attention layers followed by a linear layer [196]. The first attention layer applies self-attention over previously generated outputs, whereas the second one attends to the model conditioning in order to refine the self-attention output. The transformer model is composed of multiple transformer blocks followed by a linear layer and a softmax nonlinearity that provides a distribution over recipe words for each time step t . Figure 9.2(a) illustrates the transformer model, which traditionally is conditioned on a single modality. However, our recipe generator is conditioned on two sources: image features $\mathbf{e}_I \in \mathbb{R}^{P \times d_e}$ and ingredients embeddings $\mathbf{e}_L \in \mathbb{R}^{K \times d_e}$ (P and K denote the number of image and ingredient features, respectively, and d_e is the embedding dimensionality). Thus, we want our attention to reason about both modalities simultaneously, guiding the instruction generation process. To that end, we explore three different fusion strategies (depicted in Figure 9.2):

- **Concatenated attention.** This strategy first concatenates both image \mathbf{e}_I and ingredients \mathbf{e}_L embeddings over the first dimension $\mathbf{e}_{concat} \in \mathbb{R}^{(K+P) \times d_e}$. Then, attention is applied over the combined embeddings. Figure 9.2(b) illustrates the concatenation strategy.
- **Independent attention.** This strategy incorporates two attention layers to deal with the bi-modal conditioning. In this case, one layer attends over the image embedding \mathbf{e}_I , whereas the other attends over the ingredient embeddings \mathbf{e}_L . The output of both attention layers is combined via summation operation. Figure 9.2(c) depicts the independent attention flow.
- **Sequential attention.** This strategy sequentially attends over the two conditioning modalities. In our design, we consider two orderings: (1) *image first* where the attention is first computed over image embeddings \mathbf{e}_I and then over ingredient embeddings \mathbf{e}_L ; and (2) *ingredients first* where the order is flipped and we first attend over ingredient embeddings \mathbf{e}_L followed by image embeddings \mathbf{e}_I . Figure 9.2(d) shows the sequential attention module.

Ingredient Decoder

Which is the best structure to represent ingredients? On the one hand, it seems clear that ingredients are a *set*, since permuting them does not alter the outcome of the cooking recipe. On the other hand, we colloquially refer to ingredients as a *list* (e.g., list of ingredients), implying some order. Moreover, it would be reasonable to think that there is some information in the order in which humans write down the ingredients in a recipe. Therefore, in this subsection we consider both scenarios and introduce models that work either with a list of ingredients or with a set of ingredients. Assigning a variable number of ingredients to an image can be achieved with multi-label image classification methods, which have been extensively studied in Chapter 5 of this thesis. For this reason, we revisit several architectures presented in Chapter 5 and evaluate them for the ingredient prediction task.

A *list of ingredients* is a variable sized, ordered collection of unique meal constituents. More precisely, let us define a dictionary of ingredients of size N as $\mathcal{D} = \{d_i\}_{i=0}^N$, from which we can obtain a list of ingredients L by selecting K elements from \mathcal{D} : $L = [l_i]_{i=0}^K$. We encode L as a binary matrix \mathbf{L} of dimensions $K \times N$, with $\mathbf{L}_{i,j} = 1$ if $d_j \in \mathcal{D}$ is selected and 0 otherwise (one-hot-code representation). Thus, our training data consists of M image and ingredient list pairs $\{(\mathbf{x}^{(i)}, \mathbf{L}^{(i)})\}_{i=0}^M$. In this scenario, the goal is to predict $\hat{\mathbf{L}}$ from an image \mathbf{x} by maximizing the following objective:

$$\arg \max_{\theta_I, \theta_L} \sum_{i=0}^M \log p(\hat{\mathbf{L}}^{(i)} = \mathbf{L}^{(i)} | \mathbf{x}^{(i)}; \theta_I, \theta_L), \quad (9.1)$$

where θ_I and θ_L represent the learnable parameters of the image encoder and ingredient decoder, respectively. Since \mathbf{L} denotes a list, we can factorize $p(\hat{\mathbf{L}}^{(i)} = \mathbf{L}^{(i)} | \mathbf{x}^{(i)})$ into K conditionals: $\sum_{k=0}^K \log p(\hat{\mathbf{L}}_k^{(i)} = \mathbf{L}_k^{(i)} | \mathbf{x}^{(i)}, \mathbf{L}_{<k}^{(i)})$ ¹ and parametrize $p(\hat{\mathbf{L}}_k^{(i)} | \mathbf{x}^{(i)}, \mathbf{L}_{<k}^{(i)})$ as a categorical distribution. We model this distribution with the Transformer auto-regressive decoder (TF) presented in Chapter 5. Note that, for simplicity, auto-regressive LSTM decoders are not considered in this chapter since they achieve comparable performance to Transformer-based ones, but require more parameters. However, as highlighted in Chapter 5, this formulation inherently penalizes for ingredient order, which might not necessarily be relevant to predict ingredients.

On the other hand, a *set of ingredients* is a variable sized, unordered collection of unique meal constituents. We can obtain a set of ingredients S by selecting K ingredients from the dictionary \mathcal{D} : $S = \{s_i\}_{i=0}^K$. We represent S as a binary vector \mathbf{s} of dimension N , where $\mathbf{s}_i = 1$ if $s_i \in S$ and 0 otherwise. Thus, our training data consists of M image and ingredient set pairs: $\{(\mathbf{x}^{(i)}, \mathbf{s}^{(i)})\}_{i=0}^M$. In this case, the goal is to predict $\hat{\mathbf{s}}$ from an image \mathbf{x} by maximizing the following objective:

$$\arg \max_{\theta_I, \theta_L} \sum_{i=0}^M \log p(\hat{\mathbf{s}}^{(i)} = \mathbf{s}^{(i)} | \mathbf{x}^{(i)}; \theta_I, \theta_L). \quad (9.2)$$

Assuming independence among set elements, we can factorize:

$$p(\hat{\mathbf{s}}^{(i)} = \mathbf{s}^{(i)} | \mathbf{x}^{(i)}) = \sum_{j=0}^N \log p(\hat{\mathbf{s}}_j^{(i)} = \mathbf{s}_j^{(i)} | \mathbf{x}^{(i)}) \quad (9.3)$$

¹ $< k$ denotes all elements up to, but not including, k

In this case, we revisit models from Chapter 5 that model this formulation, namely FF_{BCE} and its variant that incorporates cardinality prediction $\text{FF}_{\text{BCE,DC}}$. However, the ingredients in the set are not necessarily independent, e.g., *salt* and *pepper* frequently appear together. For this reason, we also consider fully connected models trained with loss functions that account for ingredient co-occurrences instead of treating ingredients independently, namely FF_{TD} and FF_{sIoU} . We also consider to model element dependencies with the auto-regressive model TF_{set} , which is optimized with a permutation-invariant loss function. Note that most feed-forward methods incorporating cardinality ($\text{FF}_{\text{BCE,C}}$, $\text{FF}_{\text{sIoU,C}}$ and $\text{FF}_{\text{TD,C}}$) have not been considered in this study, since their gain w.r.t. the same architectures without cardinality prediction was marginal in many cases.

Optimization

We train our recipe transformer in two stages. In the first stage, we pre-train the image encoder and ingredients decoder as presented in Subsection 9.1.2. Then, in the second stage, we train the ingredient encoder and instruction decoder (following Subsection 9.1.1) by minimizing the negative log-likelihood and adjusting θ_R and θ_E . Note that, while training, the instruction decoder takes as input the ground truth ingredients. All transformer models are trained with teacher forcing [211] except for the set transformer.

Experiments

This section is devoted to the dataset and the description of implementation details, followed by an exhaustive analysis of the proposed attention strategies for the cooking instruction transformer. Further, we quantitatively compare the proposed ingredient prediction models to previously introduced baselines. Finally, a comparison of our inverse cooking system with retrieval-based models as well as a comprehensive user study is provided.

Dataset preprocessing

We train and evaluate our models on the Recipe1M dataset (described in Section 7.2). Since the dataset was obtained by scraping cooking websites, the resulting recipes are highly unstructured and contain frequently redundant or very narrowly defined cooking ingredients (e.g., *olive oil*, *virgin olive oil* and *spanish olive oil* are separate ingredients). Moreover, the ingredient vocabulary contains more than 400 different types of *cheese*, and more than 300 types of *pepper*. As a result, the original dataset contains 16 823 unique ingredients, which we preprocess to reduce its size and complexity. First, we merge ingredients if they share the first or last two words (e.g., *bacon cheddar cheese* is merged into *cheddar cheese*); then, we cluster the ingredients that have same word in the first or in the last position (e.g., *gorgonzola cheese* or *cheese blend* are clustered together into the *cheese* category); finally we remove plurals and discard ingredients that appear less than 10 times in the dataset. Altogether, we reduce the ingredient vocabulary from over 16k to 1 488 unique ingredients.

For the cooking instructions, we tokenize the raw text and remove words that appear less than 10 times in the dataset, and replace them with unknown word token. Moreover, we add special tokens for the start and the end of recipe as well as the end of instruction (indicating the end of a sentence in a recipe). This process results in a recipe vocabulary

of 25 828 unique words.

Implementation Details

We resize images to 256 pixels in their shortest side and take random crops of 224×224 for training and we select central 224×224 pixels for evaluation. During training we randomly flip ($p = 0.5$), rotate (± 10 degrees) and translate images ($\pm 10\%$ image size on each axis) for augmentation.

Ingredient Prediction. Feed-forward models FF_{BCE} , FF_{TD} and FF_{sIoU} were trained with a mini-batch size of 300, whereas $\text{FF}_{\text{BCE,DC}}$ was trained with a mini-batch size of 256. All of them were trained with a learning rate of 0.001. The learning rate for pre-trained ResNet layers was scaled for each model as follows: $0.01\times$ for FF_{BCE} , FF_{sIoU} and $\text{FF}_{\text{BCE,DC}}$ and $0.1\times$ for FF_{TD} . Transformer list-based models (TF) were trained with mini-batch size 300 and learning rate 0.001, scaling the learning rate of ResNet layers with a factor of $0.1\times$. Similarly, the set transformer TF_{set} was trained with mini-batch size of 300 and a learning rate of 0.0001, scaling the learning rate of pre-trained ResNet layers with a factor of $1.0\times$. The optimization of TF_{set} minimizes a cost function composed of three terms, namely the ingredient prediction loss $\mathcal{L}_{\text{ingr}}$ and the end-of-sequence loss \mathcal{L}_{eos} and the cardinality penalty $\mathcal{L}_{\text{card}}$. We set the contribution of each term with weights 1000.0 and 1.0 and 1.0, respectively. We use a label smoothing factor of 0.1 for all models trained with BCE loss (FF_{BCE} , $\text{FF}_{\text{BCE,DC}}$, TF_{set}), which we found experimentally useful.

Instruction Generation. For the instruction decoder, we use a transformer with 8 blocks and 16 multi-head attentions, each one with dimensionality 32. For the ingredient decoder, we use a transformer with 2 blocks and 4 multi-head attentions, each one with dimensionality of 128. To obtain image embeddings we use the last convolutional layer of ResNet-50 model. Both image and ingredients embeddings are of dimension 512. We use a batch size of 256 and learning rate of 0.001. Parameters of the image encoder module are taken from the ingredient prediction model and frozen during training for instruction generation.

We keep a maximum of 20 ingredients per recipe and truncate instructions to a maximum of 150 words both for training and evaluation.

All models are trained with Adam optimizer [99] ($\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 1e-8$) [99], exponential decay of 0.99 after each epoch, dropout probability 0.3 and a maximum number of 400 epochs, or until early-stopping criteria is met (using patience of 50 and monitoring validation loss). All models are implemented with PyTorch ² [155].

Recipe Generation

In this section, we compare the proposed multi-modal attention architectures described in Section 9.1.1. Table 9.1 (left) reports the results in terms of perplexity on the validation set. We observe that independent attention exhibits the lowest results, followed by both sequential attentions. While the latter have the capability to refine the output with either ingredient or image information consecutively, independent attention can only do it in one step. This is also the case of concatenated attention, which achieves the best performance. However, concatenated attention is flexible enough to decide whether

²<http://pytorch.org/>

Model	ppl	Model	IoU	F1
Independent	8.59	FF_{BCE}	17.85	30.30
Seq. img. first	8.53	FF_{sIoU}	26.25	41.58
Seq. ing. first	8.61	$FF_{BCE,DC}$	27.22	42.80
Concatenated	8.50	FF_{TD}	28.84	44.11
		TF	29.48	45.55
		TF + shuffle	27.86	43.58
		TF_{set}	31.80	48.26

Table 9.1: Model selection (validation set). Left: Recipe perplexity (ppl). Right: Global ingredient IoU & F1 metrics.

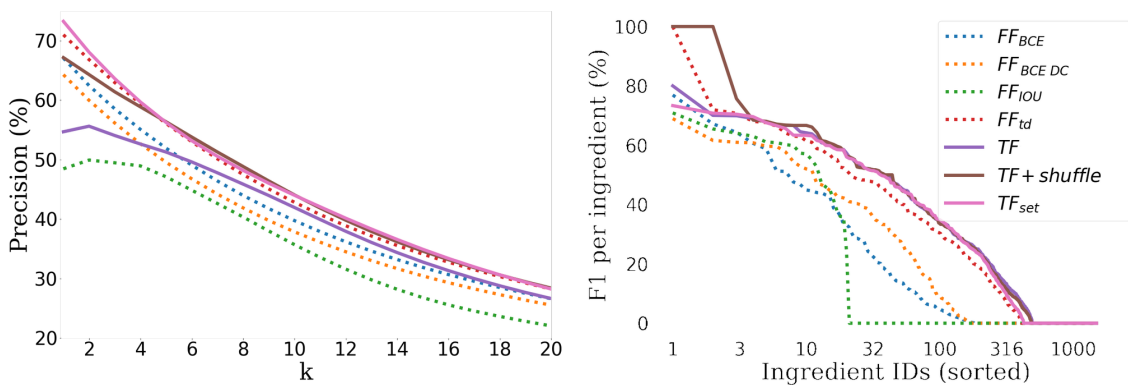


Figure 9.3: Ingredient prediction results: P@K and F1 per ingredient.

to give more focus to one modality, at the expense of the other, whereas independent attention is forced to include information from both modalities. Therefore, we use the concatenated attention model to report results on the test set. We compare it to a system going directly from image-to-sequence of instructions without predicting ingredients (I2R). Moreover, to assess the influence of visual features on recipe quality, we adapt our model by removing visual features and predicting instructions directly from ingredients (L2R). Our system achieves a test set perplexity of 8.51, improving both I2R and L2R baselines, and highlighting the benefits of using both image and ingredients when generating recipes. L2R surpasses I2R with a perplexity of 8.67 vs. 9.66, demonstrating the usefulness of having access to concepts (ingredients) that are essential to the cooking instructions. Finally, we greedily sample instructions from our model and analyze the results. We notice that generated instructions have an average of 9.21 sentences containing 9 words each, whereas real, ground truth instructions have an average of 9.08 sentences of length 12.79.

Ingredient Prediction

In this section, we compare the proposed ingredient prediction approaches to previously introduced models, with the goal of assessing whether ingredients should be treated as lists or sets. We consider models from the multilabel classification literature as baselines, and tune them for our purposes. On the one hand, we have models based on feed forward convolutional networks, which are trained to predict sets of ingredients. We experiment

	Card. error	# pred. ingrs
FF_{BCE}	5.67 ± 3.10	2.37 ± 1.58
$\text{FF}_{\text{BCE,DC}}$	2.68 ± 2.07	9.18 ± 2.06
FF_{sIoU}	2.46 ± 1.95	7.86 ± 1.72
FF_{TD}	3.02 ± 2.50	8.02 ± 3.24
TF	2.49 ± 2.11	7.05 ± 2.77
TF + shuffle	3.24 ± 2.50	5.06 ± 1.85
TF_{set}	2.56 ± 1.93	9.43 ± 2.35

Table 9.2: Ingredient Cardinality.

with several losses to train these models, namely binary cross-entropy, soft intersection over union as well as target distribution cross-entropy. Note that binary cross-entropy is the only one not taking into account dependencies among elements in the set. On the other hand, we have sequential models that predict lists, imposing order and exploiting dependencies among elements. Finally, we consider recently proposed set prediction models, which couple the set prediction with a cardinality prediction to determine which elements to include in the set [169]. The latter method assumes independence of elements in the set.

Table 9.1 (right) reports the results on the validation set for the state-of-the-art baselines as well as the proposed approaches. We evaluate the models in terms of Intersection over Union (IoU) and F1 score, computed for accumulated counts of TP , FN and FP over the entire dataset split (following Pascal VOC convention). As shown in the table, the feed forward model trained with binary cross-entropy [29] (FF_{BCE}) exhibits the lowest performance on both metrics, which could be explained by the assumed independence among ingredients. These results are already notably improved by the method that learns to predict the set cardinality ($\text{FF}_{\text{BCE,DC}}$). Similarly, the performance increases when training the model with structured losses such as soft IoU (FF_{sIoU}). Our feed forward model trained with target distribution (FF_{TD}) and sampled by thresholding ($\text{th} = 0.5$) the sum of probabilities of selected ingredients outperforms all feed forward baselines, including recently proposed alternatives for set prediction such as [169] ($\text{FF}_{\text{BCE,DC}}$). Note that target distribution models dependencies among elements in a set and implicitly captures cardinality information. Following recent literature modeling sets as lists [147], we train a transformer network to predict ingredients given an image by minimizing the negative log-likelihood loss (TF). Moreover, we train the same transformer by randomly shuffling the ingredients (thus, removing order from the data). Both models exhibit competitive results when compared to feed forward models, highlighting the importance of modeling dependencies among ingredients. Finally, our proposed set transformer TF_{set} , which models ingredient co-occurrences exploiting the auto-regressive nature of the model yet satisfying order invariance, achieves the best results, emphasizing the importance of modeling dependencies, while not penalizing for any given order.

The average number of ingredients per sample in Recipe1M is 7.99 ± 3.21 after pre-processing. We report the cardinality prediction errors as well as the average number of predicted ingredients for each of the tested models in Table 9.2. TF_{set} is the third best method in terms of cardinality error (after FF_{sIoU} and TF), while being superior to all methods in terms of F1 and IoU. Further, Figure 9.3 (left) shows the precision score at

	IoU	F1			
R_{I2L}	18.92	31.83			
R_{I2LR}	19.85	33.13			
FF_{TD} (ours)	29.82	45.94	R_{IL2R}	31.92	28.94
TF_{set} (ours)	32.11	48.61	Ours	75.47	77.13

Table 9.3: Test performance against retrieval. Left: Global ingredient IoU and F1 scores. Right: Precision and Recall of ingredients in cooking instructions.

	IoU	F1		Success %
Human	21.36	35.20	Real	80.33
Retrieved	18.03	30.55	Retrieved	48.81
Ours	32.52	49.08	Ours	55.47

Table 9.4: User studies. Left: IoU & F1 scores for ingredients obtained by retrieval system, our approach and humans. Right: Recipe success rate according to human judgment.

different values of K . As observed, the plot follows similar trends as Table 9.1 (right), with FF_{TD} being among the most competitive models and TF_{set} outperforming all previous baselines for most values of K . Figure 9.3 (right) shows the F1 per ingredient, where the ingredients in the horizontal axes are sorted by score. Again, we see that models that exploit dependencies consistently improve ingredient’s F1 scores, strengthening the importance of modeling ingredient co-occurrences.

Generation vs Retrieval

Ingredient prediction evaluation. We use the retrieval model from Chapter 8 as a baseline and compare it with our best ingredient predictions models, namely FF_{TD} and TF_{set} . The retrieval model, which we refer to as R_{I2LR} , learns joint embeddings of images and recipes (title, ingredients and instructions). Therefore, for the ingredient prediction task, we use the image embeddings to retrieve the closest recipe and report metrics for the ingredients of the retrieved recipe. We further consider an alternative retrieval architecture, which learns joint embeddings between images and ingredients list (ignoring title and instructions). We refer to this model as R_{I2L} . Table 9.3 (left) reports the obtained results on the Recipe1M test set. The R_{I2LR} model outperforms the R_{I2L} one, which indicates that instructions contain complementary information that is useful when learning effective embeddings. Furthermore, both of our proposed methods outperform the retrieval-baselines by a significant margin (e.g., TF_{set} outperforms the R_{I2LR} retrieval baseline by 12.26 IoU points and 15.48 F1 score points), which demonstrates the superiority of our models. Finally, Figure 9.4 presents some qualitative results for image-to-ingredient prediction for our model as well as for the retrieval based system. We use blue to highlight the ingredients that are present in the ground truth annotation and red otherwise. Interestingly, ingredients predicted by our model often seem plausible.

Recipe generation evaluation. We compare our proposed instruction decoder (which generates instructions given an image and ingredients) with a retrieval variant. For a fair comparison, we retrain the retrieval system to find the cooking instructions given both image and ingredients. In our evaluation, we consider the ground truth ingredients





	Ours	Retrieved	Real
	cheese onion pepper soup cream salt milk butter	potato butter soup cheese onion cream corn	milk water butter potato corn cheese onion
	shrimp butter garlic zucchini pepper soy_sauce juice	lemon salt clove catfish seasoning carrot parsley	lemon zucchini oil pepper shrimp juice salt garlic parsley onion
	sugar strawberries juice water raspberries cream	tart_shell sugar cornstarch juice strawberries	butter vanilla strawberries sugar wine vinegar cream
	cheese tomato cracker broccoli muffin	cheese cracker miracle_whip lettuce tomato	muffin cheese broccoli tomato

Figure 9.4: Ingredient prediction examples. We compare obtained ingredients with our method and the retrieval baseline. Ingredients are displayed in blue if they are present in the real sample and red otherwise. Best viewed in color.

as reference and compute *recall* and *precision* w.r.t. the ingredients that appear in the obtained instructions. Thus, recall computes the percentage of ingredients in the reference that appear in the output instructions, whereas precision measures the percentage of ingredients appearing in the instructions that also appear in the reference. Table 9.3 (right) displays comparison between our model and the retrieval system. Results show that the ingredients appearing in the cooking instructions obtained with our model have better recall and precision scores than the ingredients in retrieved instructions.

User Studies

In this section, we quantify the quality of predicted ingredients and generated instructions with user studies. In the first study, we compare the performance of our model against human performance in the task of recipe generation. We randomly select 15 images from the test set, and ask users to select up to 20 distinct ingredients as well as write a recipe that would correspond with the provided image. To reduce the complexity of the task for humans, we reduced the ingredient vocabulary from 1 488 to 323, by increasing the frequency threshold from 10 to 1k. We collected answers from 31 different users, altogether collecting an average of 5.5 answers for each image. For fair comparison, we re-train our best ingredient prediction model on the reduced vocabulary of ingredients. We compute IoU and F1 ingredient scores obtained by humans, the retrieval baseline and our method. Results are included in Table 9.4 (left), underlining the complexity of the task. As shown in the table, humans outperform the retrieval baseline (F1 of 35.20% vs 30.55%, respectively). Furthermore, our method outperforms both human baseline and retrieval based systems obtaining F1 of 49.08%.

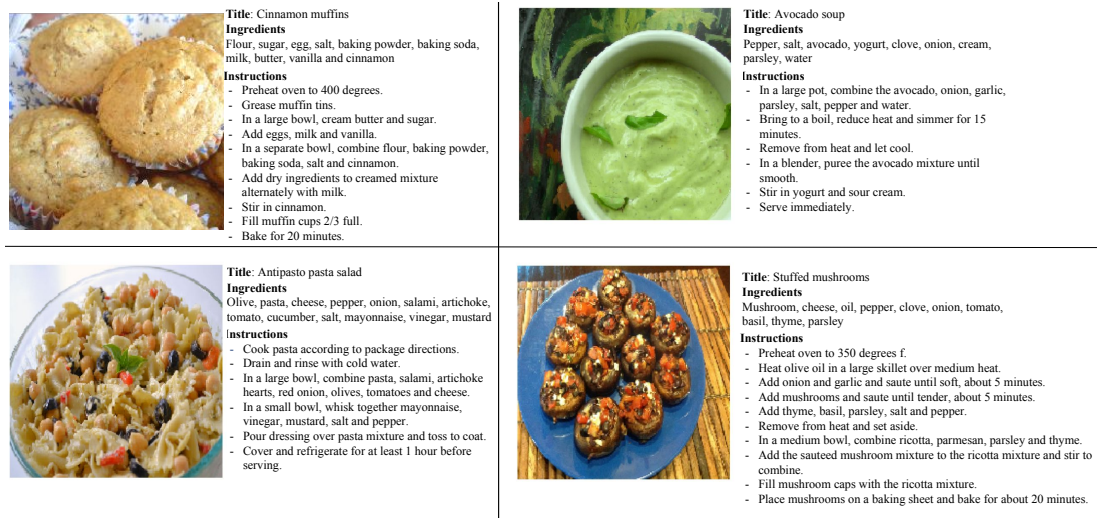


Figure 9.5: Examples of generated recipes, composed of a title, ingredients and cooking instructions.

The second study aims at quantifying the quality of the generated recipes (ingredients and instructions) with respect to (1) the real recipes in the dataset, and (2) the ones obtained with the retrieval baseline from Chapter 8. With this purpose, we randomly select 150 recipes with their associated images from the test set and, for each image, we collect the corresponding real recipe, the top-1 retrieved recipe and our generated recipe. We present the users with 15 image-recipe pairs (randomly chosen among the real, retrieved and generated ones) asking them to indicate whether the recipe matches the image. In the study, we collected answers from 105 different users, resulting in an average of 10 responses for each image. Table 9.4 (right) presents the results of this study, reporting the success rate of each recipe type. As it can be observed, the success rate of generated recipes is higher than the success rate of retrieved recipes, stressing the benefits of our approach w.r.t. retrieval.

Examples of recipes (title, ingredients and instructions) generated with our method are included in Figure 9.5.

Conclusion

In this chapter, we introduced an image-to-recipe generation system, which takes a food image and produces a recipe consisting of a title, ingredients and sequence of cooking instructions. We first predicted sets of ingredients from food images, showing that modeling dependencies matters. Then, we explored instruction generation conditioned on images and inferred ingredients, highlighting the importance of reasoning about both modalities at the same time. Finally, user study results confirm the difficulty of the task, and demonstrate the superiority of our system against state-of-the-art image-to-recipe retrieval approaches.

Summary

The last part of the thesis presented two solutions to the task of obtaining a cooking recipe given a food image. The first one (Chapter 8) posed the problem as a retrieval task, in which the recipe is retrieved as a whole (ingredients and instructions) given the input image. A joint neural network was trained to project a recipe and image representations into the same embedding space which, once trained, allowed the retrieval of one given the other. The proposed model was evaluated against strong baselines, including human performance. The second one (Chapter 9) posed the problem as a conditional generation one, where the ingredients are first predicted for the input image, and used together with image features to condition the title and cooking instruction decoder. According to human judgment, the recipe generation pipeline from Chapter 9 provided better recipes for input images than the retrieval system from Chapter 8.

Conclusions

We have seen extraordinary progress in computer vision over the past decade. Deep neural networks have narrowed down the gap between human and computer performance on long studied computer vision problems [81]. Virtually all computer vision tasks experienced boosts in performance by re-utilizing representations from pre-trained neural networks [164]. Later on, the design and training of end-to-end neural network solutions optimized for the task in hand became the go-to approach for most computer vision tasks, given enough training data [81, 125, 167, 79]. This thesis has presented contributions within these paradigms, namely the extraction and optimization of deep features from convolutional neural networks for visual retrieval in Part I, and the end-to-end formulation of image-to-set prediction tasks with neural networks in Part II.

Rapid advances in core tasks such as image classification or object detection have opened the frontiers of computer vision to replace output categorical labels with natural language to explain the visual world. We have witnessed the first neural networks to mimic human behavior to generate image descriptions [201, 128] or answer questions about images [3, 129]. In this context, Part III of this thesis has presented two approaches to tackle the challenging problem of image-to-recipe prediction. Despite great progress, current deep learning methods (including those presented in this dissertation) are known to exploit patterns from large datasets as a sort of distant proxy to commonsense knowledge, which often times leads to the illusion of human-like performance of image captioning, question answering or, as studied in this thesis, recipe generation methods. True visual understanding algorithms in the future must involve commonsense reasoning about how the world works in order to be reliable for people to use. While recent works [223] have attempted to collect annotations for new datasets to push research towards explainable and contextualized outputs of deep neural networks, here we argue that the image-to-recipe prediction problem offers a vast ground to base research upon this direction.

Figure 9.6 illustrates the complexity of the image-to-recipe prediction task with an example in the Recipe1M dataset. Notice that the recipe starts with an instruction to heat the oven, which one may acknowledge to be a reasonable choice given that the input images shows a lasagna. Notably, the 18.58% of all the recipes in the training set of Recipe1M include some version of the *heat oven* instruction as the first sentence in the recipe. If we only consider recipes including the word *lasagna* in their title, the percentage increases to 36.88%, and further reaches 49.25% when searching over all sentences in the recipe and not only the first one. Thus, it is almost straight-forward for a neural network with enough capacity and training data to cluster lasagna-like images together and accurately instruct to preheat the oven for all of them. However, the human explanation for this instruction choice goes way beyond the simple association the neural network has learned (i.e. all lasagnas require heating the oven). First, anyone with a relative cooking experience is aware of the different cooking methods, and likely knows that lasagnas fall

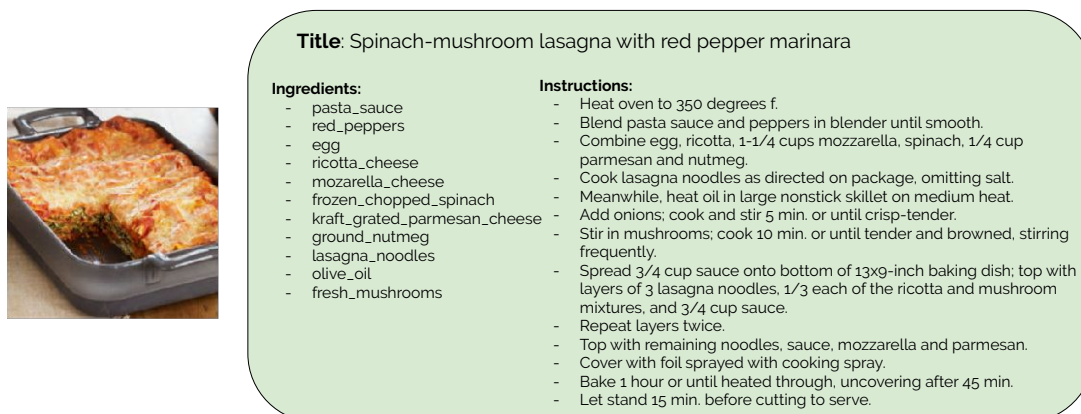


Figure 9.6: A sample in the Recipe1M dataset.

into the category of baked food, for which an oven is required. We also know that food cooks better in a warm oven (we might have experienced this ourselves, or we might have learned this from someone else). Second, humans understand the concept of time – cooking requires time, but some steps are faster to complete than others (e.g., baking a cake takes more time than chopping a few tomatoes). Further, we understand that different cooking instructions require different amounts of human intervention (beating an egg is a more human-involved task than boiling water or heating an oven). Finally, both from the laws of thermodynamics and our previous cooking experiences we get that warming up an oven until it reaches a certain temperature takes time. Thus, all this contextualized knowledge eventually leads to the decision of preheating the oven as one of the first steps to cook a lasagna, while more human-involved tasks such as mixing ingredients are performed in the meantime in order to minimize the wait time. In contrast, the ability of the model presented in Chapter 9 to correctly instruct to *preheat the oven* given a lasagna image does not involve any explicit reasoning, and instead relies on the complex statistical function learned from training data to map image pixels to sentences.

Beyond the previous example, in order for an intelligent system to predict a recipe from an image, it must overcome a series of other challenges, such as inferring ingredients subject to heavy deformations, occlusions or even invisibility. Understanding the raw form of ingredients and their physical properties is also crucial to generate a recipe, and the level of detail in ingredients also plays a critical role. For example, tomato sauce (referred to as *pasta_sauce* in the figure) can both be purchased in a store or made from raw tomatoes, and the choice of one over the other should significantly alter the resulting recipe. Further, the cooking process requires an understanding of ingredient quantities and food portions, which must be estimated from still images.

Altogether, challenges in the image-to-recipe generation problem align well with those in automating commonsense reasoning [46]. Actions that seem straight-forward such as the *preheat oven* example above require awareness of time, anticipation and planning. Commonsense reasoning involves taking decisions based on assumptions over incomplete data, and adapting to infrequent situations using knowledge from similar previous experiences. How to encode such types of information in a way that is understandable for a computer, and how to teach computers to acquire this knowledge from data to support their decisions are the challenges that shall drive future research in the field.

Publications

Publications derived from this thesis

Submitted

- Luis Pineda*, **Amaia Salvador***, Michal Drozdal, Adriana Romero. Elucidating image-to-set prediction: An analysis of models, losses and datasets. (Submitted).
- Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, **Amaia Salvador**, Yusuf Aytar, Ingmar Weber, Antonio Toralba. Recipe1M: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. (Submitted).

In Proceedings

- **Amaia Salvador**, Michal Drozdal, Xavier Giro-i-Nieto, Adriana Romero. Inverse Cooking: Recipe Generation from Food Images. In CVPR 2019.
- **Amaia Salvador***, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Ofli, Ingmar Weber and Antonio Torralba. Learning Cross-modal Embeddings for Cooking Recipes and Food Images. In CVPR 2017.
- **Amaia Salvador**, Xavier Giro-i-Nieto, Ferran Marques, Shin'ichi Satoh. Faster R-CNN Features for Instance Search. In CVPR Workshops 2016.
- Eva Mohedano*, **Amaia Salvador***, Kevin McGuinness, Xavier Giro-i-Nieto, Noel OConnor, Ferran Marques. Bags of Local Convolutional Features for Scalable Instance Search. In ICMR 2016.
- Vinh-Tiep Nguyen, Duy-Dinh Le, **Amaia Salvador**, Caizhi Zu, Dinh-Luan Nguyen, Minh-Triet Tran, Thanh-Ngo Duc, Duc-Anh Duong, Shin'ichi Satoh, Xavier Giro-i-Nieto. NIL-HITACHI-UIT at TRECVID 2015 instance search. In TRECVID Workshop 2015.
- Kevin McGuinness, Eva Mohedano, **Amaia Salvador**, Zhenxing Zhang, Mark Marsden, Peng Wang, Iveel Jargalsaikhan, Joseph Antony, Xavier Giro-i-Nieto, Shinichi Satoh, Noel E. OConnor, Alan Smeaton. Insight DCU at TRECVID 2015. In TRECVID Workshop 2015.

Other publications

- Carles Ventura, Miriam Bellver, Andreu Girbau, **Amaia Salvador**, Ferran Marques and Xavier Giro-i-Nieto. End-to-End Recurrent Net for Video Object Segmentation. In CVPR 2019.
- Amanda Duarte, Francisco Roldan, Miquel Tubau, Janna Escur, Santiago Pascual, **Amaia Salvador**, Eva Mohedano, Kevin McGuinness, Jordi Torres and Xavier Giro-i-Nieto. Wav2Pix: Speech-conditioned Face Generation using Generative Adversarial Networks. In ICASSP 2019.
- Eva Mohedano, **Amaia Salvador**, Kevin McGuinness, Xavier Giro-i-Nieto, Noel O'Connor, Ferran Marques. Object Retrieval with Deep Convolutional Features. In Deep Learning for Image Processing Applications 2017 (Book Chapter).
- Alberto Montes, **Amaia Salvador**, Santiago Pascual and Xavier Giro-i-Nieto. Temporal Activity Detection in Untrimmed Videos with Recurrent Neural Networks. In NIPS Workshops 2016.
- Victor Campos, **Amaia Salvador**, Xavier Giro-i-Nieto and Brendan Jou. Diving Deep into Sentiment: Understanding Fine-tuned CNNs for Visual Sentiment Prediction. In ACM Multimedia Workshops 2015.
- Eva Mohedano, **Amaia Salvador**, Sergi Porta, Xavier Giro-i-Nieto, Kevin McGuinness, Graham Healy, Noel O'Connor and Alan Smeaton. Exploring EEG for Object Detection and Retrieval. In ICMR 2015.
- **Amaia Salvador**, Matthias Zeppelzauer, Daniel Manchon, Andrea Calafell and Xavier Giro-i-Nieto. Cultural Event Recognition with Visual ConvNets and Temporal Models. In CVPR Workshops 2015.
- Kevin McGuinness, Eva Mohedano, ZhenXing Zhang, Feiyan Hu, Rami Albatal, Cathal Gurrin, Noel O'Connor, Alan Smeaton, **Amaia Salvador**, Xavier Giro-i-Nieto, Carles Ventura. Insight Centre for Data Analytics (DCU) at TRECVID 2014: Instance Search and Semantic Indexing Tasks. In TRECVID Workshop 2014.
- Axel Carlier, **Amaia Salvador**, Ferran Cabezas, Xavier Giro-i-Nieto, Vincent Charvillat and Oge Marques. Assessment of Crowdsourcing and Gamification Loss in User-Assisted Object Segmentation. In MTAP 2016.
- Ferran Cabezas, Axel Carlier, **Amaia Salvador**, Xavier Giro-i-Nieto and Vincent Charvillat. Quality Control in Crowdsourced Object Segmentation. In ICIP 2015.
- Axel Carlier, **Amaia Salvador**, Xavier Giro-i-Nieto, Oge Marques and Vincent Charvillat. Click'n'Cut: Crowdsourced Interactive Segmentation with Object Candidates. In ACM Multimedia Workshops 2014.
- **Amaia Salvador**, Axel Carlier, Xavier Giro-i-Nieto, Oge Marques and Vincent Charvillat. Crowdsourced Object Segmentation with a Game. In ACM Multimedia Workshops 2013.

Bibliography

- [1] Tatiana A Amor, Saulo DS Reis, Daniel Campos, Hans J Herrmann, and José S Andrade Jr. Persistence in eye movement during visual search. *Scientific reports*, 2016. [65](#)
- [2] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *ICLR*, 2013. [90](#)
- [3] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *CVPR*, 2015. [3](#), [4](#), [90](#), [119](#)
- [4] Alessandro Antonucci, Giorgio Corani, Denis Deratani Mauá, and Sandra Gabaglio. An ensemble of bayesian networks for multilabel classification. In *IJCAI*, 2013. [54](#)
- [5] Jeremy Appleyard. Optimizing Recurrent Neural Networks in cuDNN 5. <https://devblogs.nvidia.com/optimizing-recurrent-neural-networks-cudnn-5/>, 2016. [Online; accessed 13-March-2016]. [78](#)
- [6] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *CVPR*, 2016. [22](#)
- [7] Relja Arandjelović and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *CVPR*, 2012. [20](#), [21](#), [29](#)
- [8] Anurag Arnab and Philip HS Torr. Bottom-up instance segmentation using deep higher-order crfs. In *BMVC*, 2016. [72](#), [74](#)
- [9] Anurag Arnab and Philip HS Torr. Pixelwise instance segmentation with a dynamically instantiated network. In *CVPR*, 2017. [66](#), [72](#), [74](#)
- [10] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. *ICLR*, 2017. [89](#)
- [11] Artem Babenko and Victor Lempitsky. Aggregating local deep features for image retrieval. In *ICCV*, 2015. [20](#), [21](#), [29](#), [32](#), [37](#), [40](#), [45](#)
- [12] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *ECCV*, 2014. [20](#), [21](#)
- [13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. [75](#)

- [14] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *CVPR*, 2017. [66](#), [72](#)
- [15] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 34(4):98, 2015. [47](#)
- [16] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009. [70](#)
- [17] Wei Bi and James T. Kwok. Efficient multi-label classification with many labels. In *ICML*, 2013. [54](#)
- [18] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 2017. [88](#)
- [19] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *ECCV*, 2014. [84](#)
- [20] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *ICLR*, 2018. [2](#)
- [21] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a” siamese” time delay neural network. In *NeurIPS*, 1994. [90](#)
- [22] Victor Campos, Xavier Giro-i Nieto, Brendan Jou, Jordi Torres, and Shih-Fu Chang. Sentiment concept embedding for visual affect recognition. In *Multimodal Behavior Analysis in the Wild*, pages 349–367. Elsevier, 2019. [91](#)
- [23] John Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987. [1](#)
- [24] Yue Cao, Mingsheng Long, Jianmin Wang, Qiang Yang, and Philip S Yu. Deep visual-semantic hashing for cross-modal retrieval. In *ACM SIGKDD*, 2016. [91](#)
- [25] Arantxa Casanova, Guillem Cucurull, Michal Drozdal, Adriana Romero, and Yoshua Bengio. On the iterative refinement of densely connected representation levels for semantic segmentation. In *CVPR-W*, 2018. [51](#)
- [26] Lluís Castrejon, Yusuf Aytar, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Learning aligned cross-modal representations from weakly aligned data. In *CVPR*, 2016. [97](#)
- [27] Xiaobin Chang, Tao Xiang, and Timothy M Hospedales. Scalable and effective deep cca via soft decorrelation. In *CVPR*, 2018. [90](#)
- [28] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *BMVC*, 2014. [52](#), [54](#)
- [29] Jing-Jing Chen and Chong-Wah Ngo. Deep-based ingredient recognition for cooking recipe retrieval. In *ACM Multimedia*. ACM, 2016. [52](#), [54](#), [55](#), [84](#), [85](#), [105](#), [112](#)

- [30] Jing-Jing Chen, Chong-Wah Ngo, and Tat-Seng Chua. Cross-modal recipe retrieval with rich food attributes. In *ACM Multimedia*. ACM, 2017. [84](#), [105](#)
- [31] Mei-Yun Chen, Yung-Hsiang Yang, Chia-Ju Ho, Shih-Han Wang, Shane-Ming Liu, Eugene Chang, Che-Hua Yeh, and Ming Ouhyoung. Automatic chinese food identification and quantity estimation. In *SIGGRAPH Asia 2012 Technical Briefs*, 2012. [84](#)
- [32] Shang-Fu Chen, Yi-Chen Chen, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. Order-free rnn with visual attention for multi-label classification. In *AAAI*, 2018. [53](#), [55](#), [63](#), [64](#)
- [33] Tianshui Chen, Zhouxia Wang, Guanbin Li, and Liang Lin. Recurrent attentional reinforcement learning for multi-label image recognition. In *AAAI*, 2018. [53](#), [55](#), [58](#), [64](#)
- [34] Xin Chen, Hua Zhou, and Liang Diao. Chinesefoodnet: A large-scale image dataset for chinese food recognition. *CoRR*, abs/1705.02743, 2017. [84](#)
- [35] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO captions: Data collection and evaluation server. *CoRR*, abs/1504.00325, 2015. [2](#)
- [36] Yi-Ting Chen, Xiaokai Liu, and Ming-Hsuan Yang. Multi-instance object segmentation with occlusion handling. In *CVPR*, 2015. [66](#), [72](#), [74](#)
- [37] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *AMNLP*, 2014. [13](#), [14](#)
- [38] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *ACM Conference on Image and Video Retrieval*, 2009. [4](#), [53](#), [59](#)
- [39] Ondřej Chum, James Philbin, Josef Sivic, Michael Isard, and Andrew Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, 2007. [20](#)
- [40] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. [71](#)
- [41] Bo Dai, Dahua Lin, Raquel Urtasun, and Sanja Fidler. Towards diverse and natural image descriptions via a conditional gan. *ICCV*, 2017. [87](#), [92](#)
- [42] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *ECCV*, 2016. [66](#)
- [43] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016. [65](#), [66](#)
- [44] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. [1](#), [19](#)

- [45] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José M.F. Moura, Devi Parikh, and Dhruv Batra. Visual Dialog. In *CVPR*, 2017. 2, 90
- [46] Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Commun. ACM*, 2015. 120
- [47] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. In *CVPRW*, 2017. 66, 70, 72
- [48] Krzysztof Dembczyński, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, 2010. 54
- [49] Chaorui Deng, Qi Wu, Qingyao Wu, Fuyuan Hu, Fan Lyu, and Mingkui Tan. Visual grounding via accumulated attention. In *CVPR*, 2018. 90
- [50] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014. 98
- [51] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015. 15
- [52] Ori Bar El, Ori Licht, and Netanel Yosephian. GILT: generating images from long text. *CoRR*, abs/1901.02404, 2019. 85
- [53] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 1990. 13
- [54] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge. *IJCV*, 2010. 1, 36, 53, 59, 70
- [55] Fartash Faghri, David J Fleet, Jamie Ryan Kiros, and Sanja Fidler. Vse++: Improved visual-semantic embeddings. *BMVC*, 2018. 91
- [56] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *ACL*, 2018. 87, 91
- [57] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the.. *ICLR*, 2018. 87
- [58] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008. 1, 22
- [59] Claude Fischler. Food, self and identity. *Information (International Social Science Council)*, 1988. 83
- [60] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *NeurIPS*, 2013. 91
- [61] Venkata Rama Kiran Garimella, Abdulrahman Alfayad, and Ingmar Weber. Social media image analysis for public health. In *CHI*, 2016. 83
- [62] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017. 15, 87, 91

- [63] Ross Girshick. Fast R-CNN. In *ICCV*, 2015. 2, 36, 52
- [64] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2, 36, 52
- [65] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):105, 2017. 86
- [66] Gene H Golub and Hongyuan Zha. The canonical correlations of matrix pairs and their numerical computation. In *Linear algebra for signal processing*, pages 27–49. Springer, 1995. 90
- [67] Yunchao Gong, Yangqing Jia, Thomas Leung, Alexander Toshev, and Sergey Ioffe. Deep convolutional ranking for multilabel image annotation. *CoRR*, abs/1312.4894, 2013. 53, 55, 56, 63
- [68] Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*, 2014. 22
- [69] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. 9, 14
- [70] Albert Gordo, Jon Almazan, Jerome Revaud, and Diane Larlus. Deep image retrieval: Learning global representations for image search. In *ECCV*, 2016. 47, 93
- [71] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *CVPR*, 2017. 2, 3, 4
- [72] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. 8
- [73] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, 2014. 52
- [74] Kristian J. Hammond. CHEF: A model of case-based planning. In *AAAI*. Morgan Kaufmann, 1986. 84
- [75] Xiaobing Han, Yanfei Zhong, Liqin Cao, and Liangpei Zhang. Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification. *Remote Sensing*, 2017. 11
- [76] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 70
- [77] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *ECCV*, 2014. 65, 66, 72, 74
- [78] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015. 65, 66

- [79] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. [2](#), [3](#), [51](#), [52](#), [65](#), [66](#), [72](#), [119](#)
- [80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *TPAMI*, 2015. [36](#)
- [81] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. [2](#), [10](#), [51](#), [54](#), [56](#), [67](#), [83](#), [92](#), [95](#), [106](#), [119](#)
- [82] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997. [13](#), [57](#)
- [83] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. *ECCV*, 2012. [74](#)
- [84] Seunghoon Hong, Dingdong Yang, Jongwook Choi, and Honglak Lee. Inferring semantic layout for hierarchical text-to-image synthesis. In *CVPR*, 2018. [90](#)
- [85] Daniel J Hsu, Sham M Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NeurIPS*. Curran Associates, Inc., 2009. [54](#)
- [86] Qiuyuan Huang, Zhe Gan, Asli Çelikyilmaz, Dapeng Oliver Wu, Jianfeng Wang, and Xiaodong He. Hierarchically structured reinforcement learning for topically coherent visual story generation. *CoRR*, abs/1805.08191, 2018. [92](#)
- [87] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, pages 240–248, 2016. [60](#)
- [88] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Improving bag-of-features for large scale image search. *IJCV*, 2010. [20](#)
- [89] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010. [21](#), [22](#), [26](#)
- [90] Simon Jégou, Michal Drozdal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisù: Fully convolutional densenets for semantic segmentation. In *CVPR-W*, 2017. [51](#), [83](#)
- [91] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, 2014. [28](#)
- [92] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017. [3](#), [4](#)
- [93] Yannis Kalantidis, Clayton Mellina, and Simon Osindero. Cross-dimensional weighting for aggregated deep convolutional features. In *ECCV*, 2016. [20](#), [22](#), [26](#), [29](#), [32](#), [33](#), [34](#), [37](#), [40](#), [44](#), [45](#)
- [94] Ashish Kapoor, Raajay Viswanathan, and Prateek Jain. Multilabel classification using bayesian compressed sensing. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NeurIPS*. Curran Associates, Inc., 2012. [54](#)

- [95] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, 2015. [91](#), [92](#), [93](#), [98](#)
- [96] Andrej Karpathy, Armand Joulin, and Li F Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. In *NeurIPS*, 2014. [91](#)
- [97] Yoshiyuki Kawano and Keiji Yanai. Foodcam: A real-time food recognition system on a smartphone. *MTAP*, 2015. [84](#)
- [98] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. Globally coherent text generation with neural checklist models. In *EMNLP*. Association for Computational Linguistics, 2016. [84](#)
- [99] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014. [8](#), [60](#), [71](#), [110](#)
- [100] R. Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors. In *NeurIPS*, 2015. [89](#), [94](#), [98](#)
- [101] Jonathan Krause, Justin Johnson, Ranjay Krishna, and Li Fei-Fei. A hierarchical approach for generating descriptive image paragraphs. In *CVPR*, 2017. [3](#), [87](#), [92](#)
- [102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012. [1](#), [2](#), [11](#), [36](#), [51](#), [52](#), [92](#)
- [103] Tomasz Kusmierczyk, Christoph Trattner, and Kjetil Norvag. Understanding and predicting online food recipe production patterns. In *HyperText*, 2016. [85](#)
- [104] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. [27](#)
- [105] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014. [98](#)
- [106] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. [1](#)
- [107] Peter Lennie. The cost of cortical computation. *Current biology*, 2003. [25](#)
- [108] Liang Li, Shuhui Wang, Shuqiang Jiang, and Qingming Huang. Attentive recurrent neural network for weak-supervised multi-label image classification. In *ACM Multimedia*, 2018. [53](#), [54](#), [63](#), [64](#)
- [109] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18(185):1–52, 2018. [54](#), [60](#)
- [110] Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J Guibas. Joint embeddings of shapes and images via cnn image purification. *ACM Transactions on Graphics (TOG)*, 34(6):234, 2015. [93](#)
- [111] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, 2017. [65](#)

- [112] Yuncheng Li, Yale Song, and Jiebo Luo. Improving pairwise ranking for multi-label image classification. In *CVPR*, 2017. [53](#), [55](#), [63](#), [64](#)
- [113] Xiaodan Liang, Liang Lin, Yunchao Wei, Xiaohui Shen, Jianchao Yang, and Shuicheng Yan. Proposal-free network for instance-level object segmentation. *TPAMI*, 2018. [72](#), [74](#)
- [114] Xiaodan Liang, Yunchao Wei, Xiaohui Shen, Zequn Jie, Jiashi Feng, Liang Lin, and Shuicheng Yan. Reversible recursive instance-level object segmentation. In *CVPR*, 2016. [65](#), [66](#), [70](#), [72](#), [74](#)
- [115] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. [36](#)
- [116] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. [2](#), [3](#), [4](#), [40](#), [51](#), [53](#), [59](#), [90](#)
- [117] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. Multi-label classification via feature-aware implicit label space encoding. In *ICML*, 2014. [55](#)
- [118] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, and Yunsheng Ma. Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment. In *ICOST*, 2016. [84](#)
- [119] Feng Liu, Tao Xiang, Timothy M Hospedales, Wankou Yang, and Changyin Sun. Semantic regularisation for recurrent image annotation. In *CVPR*, 2017. [53](#), [54](#), [63](#), [64](#)
- [120] Shu Liu, Xiaojuan Qi, Jianping Shi, Hong Zhang, and Jiaya Jia. Multi-scale patch aggregation (mpa) for simultaneous detection and segmentation. In *CVPR*, 2016. [70](#), [74](#)
- [121] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. [36](#)
- [122] Weiwei Liu and Ivor W. Tsang. Large margin metric learning for multi-label prediction. In *AAAI*, 2015. [54](#)
- [123] Yongcheng Liu, Lu Sheng, Jing Shao, Junjie Yan, Shiming Xiang, and Chunhong Pan. Multi-label image classification via knowledge distillation from weakly-supervised detection. In *ACM Multimedia*, 2018. [54](#), [63](#), [64](#)
- [124] Yu Liu, Yanming Guo, Song Wu, and Michael S Lew. Deepindex for accurate and efficient image retrieval. In *ICMR*, 2015. [21](#), [22](#)
- [125] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. [2](#), [12](#), [51](#), [66](#), [67](#), [83](#), [119](#)
- [126] David G Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial intelligence*, 31(3):355–395, 1987. [1](#)
- [127] David G Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999. [1](#), [2](#), [19](#), [21](#)

- [128] Jiasen Lu, Caiming Xiong, Devi Parikh, and Richard Socher. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *CVPR*, 2017. [3](#), [5](#), [15](#), [57](#), [58](#), [90](#), [92](#), [119](#)
- [129] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *NeurIPS*, 2016. [3](#), [119](#)
- [130] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. *CoRR*, abs/1805.00932, 2018. [53](#), [55](#), [56](#)
- [131] Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Generating images from captions with attention. *ICLR*, 2016. [90](#)
- [132] David Marr. *Vision*. W. H. Freeman and Company, 1982. [1](#)
- [133] Sara McGuire. Food Photo Frenzy: Inside the Instagram Craze and Travel Trend. <https://www.business.com/articles/food-photo-frenzy-inside-the-instagram-craze-and-travel-trend/>, 2017. [Online; accessed Nov-2018]. [83](#)
- [134] Tao Mei, Yong Rui, Shipeng Li, and Qi Tian. Multimedia search reranking: A literature survey. *ACM Computing Surveys (CSUR)*, 2014. [20](#)
- [135] Yelena Mejova, Sofiane Abbar, and Hamed Haddadi. Fetishizing food in digital age: #foodporn around the world. In *ICWSM*, 2016. [83](#)
- [136] Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P Murphy. Im2calories: towards an automated mobile vision food diary. In *ICCV*, 2015. [84](#)
- [137] Simon Mezgec and Barbara Koroui Seljak. Nutrinet: A deep learning food and drink image recognition system for dietary assessment. *Nutrients*, 9(7), 2017. [84](#)
- [138] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *IJCV*, 2004. [21](#)
- [139] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. [88](#), [89](#), [94](#), [98](#), [102](#)
- [140] Weiqing Min, Bing-Kun Bao, Shuhuan Mei, Yaohui Zhu, Yong Rui, and Shuqiang Jiang. You are what you eat: Exploring rich recipe information for cross-region food analysis. *IEEE Transactions on Multimedia*, 2018. [83](#), [85](#)
- [141] Massimo Minervini, Andreas Fischbach, Hanno Scharr, and Sotirios A Tsafaris. Finely-grained annotated datasets for image-based plant phenotyping. *Pattern recognition letters*, 2016. [70](#), [71](#)
- [142] Hans Moravec. *Mind children: The future of robot and human intelligence*. Harvard University Press, 1988. [1](#)

- [143] Shinsuke Mori, Hirokuni Maeta, Tetsuro Sasada, Koichiro Yoshino, Atsushi Hashimoto, Takuya Funatomi, and Yoko Yamakata. Flowgraph2text: Automatic sentence skeleton compilation for procedural text generation. In *INLG*. The Association for Computer Linguistics, 2014. 84
- [144] Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. Flow graph corpus from recipe texts. In *LREC*, 2014. 84
- [145] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010. 6
- [146] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. Large-scale multi-label text classification — revisiting neural networks. In *ECMLPKDD*, 2014. 54
- [147] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *NeurIPS*, 2017. 54, 112
- [148] Isaac Newton. *Opticks, or, a treatise of the reflections, refractions, inflections & colours of light*. 1704. 1
- [149] Joe Ng, Fan Yang, and Larry Davis. Exploiting local features from deep networks for image retrieval. In *CVPR Workshops*, 2015. 22, 32
- [150] Vinh-Tiep Nguyen, Dinh-Luan Nguyen, Minh-Triet Tran, Duy-Dinh Le, Duc Anh Duong, and Shin’ichi Satoh. Query-adaptive late fusion with neural network for instance search. In *International Workshop on Multimedia Signal Processing (MMSP)*, 2015. 22, 25
- [151] Ferda Offi, Yusuf Aytar, Ingmar Weber, Raggi Al Hammouri, and Antonio Torralba. Is saki #delicious? the food perception gap on instagram and its relation to health. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017. 4, 84
- [152] Christopher Olah. Understanding LSTM Networks. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015. [Online; accessed 13-March-2018]. 16
- [153] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001. 19
- [154] Seymour A Papert. The summer vision project. 1966. 1
- [155] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NeurIPS-W*, 2017. 60, 110
- [156] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 88
- [157] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. Large-scale image retrieval with compressed fisher vectors. In *CVPR*, 2010. 21, 26

- [158] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007. [21](#), [23](#)
- [159] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008. [23](#)
- [160] Gillian Porter, Tom Troscianko, and Iain D Gilchrist. Effort during visual search and counting: Insights from pupillometry. *The Quarterly Journal of Experimental Psychology*, 2007. [65](#)
- [161] F. Radenović, G. Tolias, and O. Chum. CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples. In *ECCV*, 2016. [47](#)
- [162] F. Radenović, G. Tolias, and O. Chum. Fine-tuning CNN image retrieval with no human annotation. *TPAMI*, 2018. [47](#)
- [163] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. [102](#)
- [164] Ali S Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *CVPR Workshops*, 2014. [21](#), [22](#), [45](#), [119](#)
- [165] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. [36](#), [51](#), [52](#), [83](#)
- [166] Mengye Ren and Richard S. Zemel. End-to-end instance segmentation with recurrent attention. In *CVPR*, 2017. [65](#), [66](#), [67](#), [69](#), [71](#), [72](#)
- [167] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. [2](#), [35](#), [36](#), [37](#), [38](#), [51](#), [52](#), [65](#), [66](#), [83](#), [119](#)
- [168] S Hamid Rezaatofghi, Anton Milan, Ehsan Abbasnejad, Anthony Dick, Ian Reid, et al. Deepsetnet: Predicting sets with deep neural networks. In *ICCV*, 2017. [53](#), [55](#), [64](#)
- [169] S Hamid Rezaatofghi, Anton Milan, Qinfeng Shi, Anthony Dick, and Ian Reid. Joint learning of set cardinality and state distribution. *AAAI*, 2018. [53](#), [55](#), [57](#), [64](#), [112](#)
- [170] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, 1951. [8](#)
- [171] Lawrence G Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963. [1](#)
- [172] Bernardino Romera-Paredes and Philip Hilaire Sean Torr. Recurrent instance segmentation. In *ECCV*, 2016. [65](#), [66](#), [67](#), [68](#), [69](#), [71](#), [72](#)
- [173] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. [67](#)

- [174] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961. 6
- [175] Yong Rui, Thomas S Huang, Michael Ortega, and Sharad Mehrotra. Relevance feedback: a power tool for interactive content-based image retrieval. *IEEE Transactions on circuits and systems for video technology*, 1998. 19
- [176] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Neurocomputing: Foundations of research. chapter learning internal representations by error propagation, 1988. 8
- [177] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 1, 2, 10, 28, 51, 56, 67
- [178] Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. A baseline for visual instance retrieval with deep convolutional networks. In *ICLR*. ICLR, 2015. 20, 21, 22, 32, 35
- [179] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *ACL*, 2018. 4, 92
- [180] Kevin J Shih, Saurabh Singh, and Derek Hoiem. Where to look: Focus regions for visual question answering. In *CVPR*, 2016. 3
- [181] Xin Shu, Darong Lai, Huanliang Xu, and Liang Tao. Learning shared subspace for multi-label dimensionality reduction via dependence maximization. *Neurocomputing*, 2015. 54
- [182] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2, 10, 28, 39, 51, 54, 83, 92, 95
- [183] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *TPAMI*, 2009. 21, 25
- [184] Alan F. Smeaton, Paul Over, and Wessel Kraaij. Evaluation campaigns and trecvid. In *International Workshop on Multimedia Information Retrieval (MIR)*, 2006. 23
- [185] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *TPAMI*, 2000. 19
- [186] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *NeurIPS*, 2013. 91
- [187] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013. 89
- [188] Yale Song, Daniel McDuff, Deepak Vasisht, and Ashish Kapoor. Exploiting sparsity and co-occurrence structure for action unit recognition. In *FG*. IEEE Computer Society, 2015. 54

- [189] Russell Stewart, Mykhaylo Andriluka, and Andrew Y Ng. End-to-end people detection in crowded scenes. In *CVPR*, 2016. 65
- [190] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NeurIPS*, 2014. 5, 15, 52, 87, 89, 91, 94
- [191] Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991. 19
- [192] Giorgos Tolias, Ronan Sifre, and Hervé Jégou. Particular object retrieval with integral max-pooling of cnn activations. In *ICLR*, 2016. 20, 21, 22, 26, 30, 32, 35, 37, 44, 45
- [193] Grigorios Tsoumikas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *ECML*, 2007. 54
- [194] Panu Turcot and D Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems. In *ICCV Workshops*, 2009. 29
- [195] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *IJCV*, 2013. 36
- [196] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 5, 15, 16, 57, 58, 87, 91, 106, 107
- [197] William E Vinje and Jack L Gallant. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science*, 2000. 25
- [198] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *ICLR*, 2015. 105
- [199] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016. 67
- [200] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NeurIPS*, 2015. 67
- [201] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *CVPR*, 2015. 2, 3, 15, 65, 90, 92, 98, 119
- [202] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001. 1, 2
- [203] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. CNN-RNN: A unified framework for multi-label image classification. In *CVPR*, 2016. 53, 54, 55, 58, 64
- [204] Xin Wang, Zhiqiang Hou, Wangsheng Yu, Yang Xue, Zefenfen Jin, and Bo Dai. Robust visual tracking via multiscale deep sparse networks. *Optical Engineering*, 56(4):043107, 2017. 7

- [205] Xin Wang, Devinder Kumar, Nicolas Thome, Matthieu Cord, and Frédéric Precioso. Recipe recognition with large multimodal food dataset. In *ICME Workshops*, 2015. [84](#), [105](#)
- [206] Zhe Wang, Wei He, Hua Wu, Haiyang Wu, Wei Li, Haifeng Wang, and Enhong Chen. Chinese poetry generation with planning based neural network. *CoRR*, abs/1610.09889, 2016. [87](#)
- [207] Zhouxia Wang, Tianshui Chen, Guanbin Li, Ruijia Xu, and Liang Lin. Multi-label image recognition by recurrently discovering attentional regions. In *ICCV*, 2017. [53](#), [55](#), [64](#)
- [208] Yunchao Wei, Wei Xia, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao, and Shuicheng Yan. CNN: single-label to multi-label. *CoRR*, abs/1406.5726, 2014. [53](#), [55](#)
- [209] Yunchao Wei, Wei Xia, Min Lin, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao, and Shuicheng Yan. Hcp: A flexible cnn framework for multi-label image classification. *TPAMI*, 2016. [53](#), [54](#)
- [210] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011. [55](#)
- [211] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1989. [109](#)
- [212] Baoyuan Wu, Fan Jia, Wei Liu, Bernard Ghanem, and Siwei Lyu. Multi-label learning with missing labels using mixed dependency graphs. *IJCV*, 2018. [54](#)
- [213] Lingxi Xie, Q Tian, R Hong, and B Zhang. Image classification and retrieval are one. In *ICMR*, 2015. [20](#)
- [214] SHI Xingjian, Zhouong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation. In *NeurIPS*, 2015. [66](#), [68](#)
- [215] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015. [2](#), [3](#), [15](#), [90](#), [92](#)
- [216] Ruihan Xu, Luis Herranz, Shuqiang Jiang, Shuang Wang, Xinhang Song, and Ramesh Jain. Geolocalized modeling for dish recognition. *IEEE Transactions on Multimedia*, 2015. [84](#)
- [217] Keiji Yanai and Yoshiyuki Kawano. Food image recognition using deep convolutional network with pre-training and fine-tuning. In *ICMEW*, 2015. [84](#)
- [218] Hao Yang, Joey Tianyi Zhou, Yu Zhang, Bin-Bin Gao, Jianxin Wu, and Jianfei Cai. Exploit bounding box annotations for multi-label object recognition. In *CVPR*, 2016. [53](#), [54](#)
- [219] Shulin Yang, Mei Chen, Dean Pomerleau, and Rahul Sukthankar. Food recognition using statistics of pairwise local features. In *CVPR*, 2010. [84](#)

- [220] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. Learning deep latent spaces for multi-label classification. *CoRR*, abs/1707.00418, 2017. [55](#)
- [221] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017. [87](#)
- [222] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. [7](#), [39](#)
- [223] Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. From recognition to cognition: Visual commonsense reasoning. *CVPR*, 2019. [3](#), [4](#), [119](#)
- [224] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017. [2](#), [90](#)
- [225] Junjie Zhang, Qi Wu, Chunhua Shen, Jian Zhang, and Jianfeng Lu. Multilabel image classification with regional latent semantic dependencies. *IEEE Transactions on Multimedia*, 2018. [53](#), [55](#), [63](#), [64](#)
- [226] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recogn.*, 40(7), July 2007. [54](#)
- [227] Wei Zhang and Chong-Wah Ngo. Searching visual instances with topology checking and context modeling. In *ICMR*, 2013. [21](#)
- [228] Wei Zhang and Chong-Wah Ngo. Topological spatial verification for instance search. *IEEE Transactions on Multimedia*, 2015. [20](#), [22](#), [25](#)
- [229] Yimeng Zhang, Zhaoyin Jia, and Tsuhan Chen. Image retrieval with geometry-preserving visual phrases. In *CVPR*, 2011. [20](#)
- [230] Feipeng Zhao and Yuhong Guo. Semi-supervised multi-label learning with incomplete labels. In *IJCAI*, 2015. [54](#)
- [231] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene CNNs. *ICLR*, 2015. [102](#)
- [232] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. [53](#), [59](#)
- [233] Xiao Zhou, Cai-Zhi Zhu, Qiang Zhu, Shinichi Satoh, and Yu-Tang Guo. A practical spatial re-ranking method for instance search from videos. In *ICIP*, 2014. [22](#), [25](#)
- [234] Cai-Zhi Zhu, Hervé Jégou, and Shinichi Satoh. Query-adaptive asymmetrical dissimilarities for visual object retrieval. In *ICCV*, 2013. [21](#)
- [235] Cai-Zhi Zhu and Shin'ichi Satoh. Large vocabulary quantization for searching instances from videos. In *ICMR*, 2012. [21](#), [25](#)
- [236] Feng Zhu, Hongsheng Li, Wanli Ouyang, Nenghai Yu, and Xiaogang Wang. Learning spatial regularization with image-level supervisions for multi-label image classification. In *CVPR*, 2017. [52](#), [53](#), [54](#), [55](#), [63](#), [64](#)

- [237] Moshe M. Zloof. Query-by-example: A data base language. *IBM systems Journal*, 1977. [19](#)