

CONNECTING IMAGES AND NATURAL LANGUAGE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Andrej Karpathy
August 2016

© 2016 by Andrej Karpathy. All Rights Reserved.

Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.

<http://creativecommons.org/licenses/by-nc/3.0/us/>

This dissertation is online at: <http://purl.stanford.edu/wf528qt3314>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Fei-Fei Li, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Percy Liang

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christopher Manning

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

A long-standing goal in the field of artificial intelligence is to develop agents that can perceive and understand the rich visual world around us and who can communicate with us about it in natural language. Significant strides have been made towards this goal over the last few years due to simultaneous advances in computing infrastructure, data gathering and algorithms. The progress has been especially rapid in visual recognition, where computers can now classify images into categories with a performance that rivals that of humans, or even surpasses it in some cases such as classifying breeds of dogs. However, despite much encouraging progress, most of the advances in visual recognition still take place in the context of assigning one or a few discrete labels to an image (e.g. *person, boat, keyboard, etc.*).

In this dissertation we develop models and techniques that allow us to connect the domain of visual data and the domain of natural language utterances, enabling translation between elements of the two domains. In particular, first we introduce a model that embeds both images and sentences into a common multi-modal embedding space. This space then allows us to identify images that depict an arbitrary sentence description and conversely, we can identify sentences that describe any image. Second, we develop an image captioning model that takes an image and directly generates a sentence description without being constrained a finite collection of human-written sentences to choose from. Lastly, we describe a model that can take an image and both localize and describe all of its salient parts. We demonstrate that this model can also be used backwards to take any arbitrary description (e.g. *white tennis shoes*) and efficiently localize the described concept in a large collection of images. We argue that these models, the techniques they take advantage of internally and the interactions they enable are a stepping stone towards artificial intelligence and that connecting images and natural language offers many practical benefits and immediate valuable applications.

From the modeling perspective, instead of designing and staging explicit algorithms to process images and sentences in complex processing pipelines, our contribution lies in the design of hybrid convolutional and recurrent neural network architectures that connect visual data and natural language utterances with a single network. Therefore, the computational processing of images,

sentences, and the structure of the multimodal embeddings that associate them emerges automatically during the process of optimizing a loss function with respect to the network's parameters over training datasets of images and their captions. This approach enjoys many of the benefits of neural networks including the use of simple, homogeneous computations that are easy to parallelize on hardware, and strong performance due to end-to-end training that formulates the problem as a single optimization problem in which all components of the model share the same end objective. We show that our models advance the state of the art on tasks that require joint processing of images and natural language and that we can design the architectures in ways that facilitate interpretable visual inspection of the network's predictions.

Acknowledgments

There are many people I must thank for contributing to the five wonderful years of my experience as a PhD student.

First I must thank my adviser Fei-Fei Li who, through more than 2,000 emails and many meetings over five years, molded and chiseled me from an eager but mostly confused student to a competent researcher. Fei-Fei's passion, foresight, ambition and zeal for perfection are infectious. When I came up with an ambitious project goal she pushed me to work on the next step. When I produced terrible paper drafts she patiently repeated the same thing 10 times until I got it. When I pitched an idea she distilled its essence, uncovered the more fundamental story and placed in a wider context. I am very grateful to Fei-Fei for teaching me not just about some aspects of computer vision or the art of communicating ideas, but for teaching me how to think.

I've been very fortunate to learn from many other remarkably inspiring people who I've developed a lot of respect and admiration for and who have become my role models in many respects. I would like to especially thank Chris Manning who is a seemingly infinite generative model of unique perspectives and insightful comments, feedback and advice. My thinking and research philosophy has similarly been shaped by thoughtful discussions and interactions with Percy Liang. I would also like to thank Vladlen Koltun, Andrew Ng, Daphne Koller, Sebastian Thrun, Noah Goodman, Leo Guibas, Silvio Saverese and Gordon Wetzstein who have all shaped my approach to research, my thoughts on innovation and who inspired me to think bigger and aim higher.

I would like to thank my close collaborators who I've had the distinct pleasure of working with and learning from. This especially includes Justin Johnson who has an uncanny ability to power through code and results one week before a deadline and Stephen Miller, whose massive output and productivity are apparently not a function of the amount of time spent sleeping. It was also a great pleasure to collaborate with Adam Coates, Armand Joulin, Olga Russakovsky, Jon Krause and Richard Socher.

I have to thank many people who have contributed to my day-to-day life and who have made my experience at Stanford so pleasant. From the Computer Vision Lab: Serena Yeung, Alireza Fathi, Alexandre Alahi, Guido Pusiol, Lamberto Ballan, Catalin Iordan, Michelle Greene, Chris Baldassano, Timnit Gebru, Ranjay Krishna, Kevin Tang, Emily Tang, Vignesh Ramanathan, Yuke

Zhu, Albert Haque, Juan Carlos Niebles, Joseph Lim, Jia Deng and Bangpeng Yao.

There are many others who I was fortunate to get to know and become friends with: Ben Poole, Sida Wang, Volodymyr Kuleshov, Jake Lussier, Victoria Popic, Kevin Lewi, Jesse Levinson, David Held, Irene Kaplow, Alex Teichman, Brody Huval, Thang Luong, Ngiam Jiquan, Will Zou, Sam Bowman, Gabor Angeli, Jean Wu, Sergey Levine, Philipp Krähenbühl, Jacob Steinhardt, Tudor Achim, Bharath Ramsundar, Kai Sheng Tai, Andre Esteva, Brett Kuprel, Manolis Savva, Okke Schrijvers, Jonathan Ho and Jon Gauthier.

During my PhD I also squeezed in two summer internships at Google and one internship at DeepMind. All three were a wonderful learning experience that had a lot of impact on my research trajectory. From these internships I would especially like to thank Greg Corrado, Tom Dean, Jon Shlens, Jeff Dean, George Toderici, Sanketh Shetty, Rahul Sukthankar, Volodymyr Mnih, Koray Kavukcuoglu, David Silver, and Geoff Hinton.

I am thankful to several funding agencies that supported my research, including the Office of Naval Research Multidisciplinary University Research Initiative (ONR MURI), Intel, Yahoo! Labs, and NVIDIA.

I would also like to extend thanks to people who I had the pleasure of interacting with and learning from during my Master's degree just before I entered as a PhD student at Stanford, especially Michiel van de Panne, Stelian Coros, Nando de Freitas and Kevin Murphy.

A big thank you to my sister who always believes in me ten times more than I do. And to my parents, who sacrificed the comfort of Slovakia to move our family to Canada when I was young, and in so doing allowed me to grasp new opportunities and pursue my dreams. It is in large part my determination to vindicate their leap of faith and make them proud that drives my ambitions.

Contents

| | |
|---|-----------|
| Abstract | iv |
| Acknowledgments | vi |
| 1 Introduction | 1 |
| 1.1 Overview | 1 |
| 1.2 Related Work | 4 |
| 1.3 Contributions and Outline | 8 |
| 2 Deep Learning Background | 10 |
| 2.1 Supervised Learning | 10 |
| 2.2 Optimization | 14 |
| 2.3 Backpropagation | 16 |
| 2.4 Neural Networks | 19 |
| 2.4.1 Vanilla Neural Networks | 19 |
| 2.4.2 Convolutional Neural Networks | 20 |
| 2.4.3 Recurrent Neural Networks | 23 |
| 2.5 Summary | 26 |
| 3 Matching Images and Sentences | 28 |
| 3.1 Related Work | 28 |
| 3.2 Model | 29 |
| 3.2.1 Representing Images | 30 |
| 3.2.2 Representing Sentences | 31 |
| 3.2.3 Alignment Objective | 34 |
| 3.3 Optimization | 37 |
| 3.4 Experiments | 37 |
| 3.4.1 Data | 37 |
| 3.4.2 Ranking Evaluation | 38 |

| | | |
|----------|--|-----------|
| 3.4.3 | Qualitative Evaluation | 40 |
| 3.5 | Conclusions | 42 |
| 4 | Generating Image Captions | 43 |
| 4.1 | Related Work | 44 |
| 4.2 | Model | 44 |
| 4.3 | Optimization | 46 |
| 4.4 | Experiments | 47 |
| 4.5 | Conclusions | 51 |
| 5 | Dense Image Captioning | 52 |
| 5.1 | Related Work | 54 |
| 5.2 | Model | 54 |
| 5.2.1 | Model Architecture | 55 |
| 5.2.2 | Loss Function | 59 |
| 5.2.3 | Training and Optimization | 59 |
| 5.3 | Experiments | 59 |
| 5.3.1 | Dense Captioning | 60 |
| 5.3.2 | Image Retrieval using Regions and Captions | 63 |
| 5.4 | Conclusions | 66 |
| 6 | Conclusions | 67 |
| | Bibliography | 71 |

List of Tables

| | | |
|-----|---|----|
| 3.1 | Image-Sentence ranking experiment results. R@K is Recall@K (high is good). Med r is the median rank (low is good). In the results for our models, we take the top 5 validation set models, evaluate each independently on the test set and then report the average performance. The standard deviations on the recall values range from approximately 0.5 to 1.0. | 39 |
| 4.1 | Evaluation of full image predictions on 1,000 test images. B-n is BLEU score that uses up to n-grams. High is good in all columns. For future comparisons, our METEOR/CIDEr Flickr8K scores are 16.7/31.8 and the Flickr30K scores are 15.3/24.7. | 48 |
| 4.2 | Learned word representations. Our model automatically arranges all words in an embedding, where nearby words have similar affects on the model. This table shows some example word queries (in bold) and to the right of them their closest words in the learned embedding in order. | 48 |
| 5.1 | Dense captioning evaluation on the test set of 5,000 images. The language metric is METEOR (high is good), our dense captioning metric is Average Precision (AP, high is good), and the test runtime performance for a 720×600 image with 300 proposals is given in milliseconds on a Titan X GPU (ms, low is good). EB, RPN, and GT correspond to EdgeBoxes [112], Region Proposal Network [81], and ground truth boxes respectively, used at test time . Therefore, the numbers in GT columns (italic) serve as upper bounds assuming perfect localization. | 61 |
| 5.2 | Results for image retrieval experiments. We evaluate ranking using recall at k ($R@K$, higher is better) and median rank of the target image (Med.rank, lower is better). We evaluate localization using ground-truth region recall at different IoU thresholds ($\text{IoU}@t$, higher is better) and median IoU (Med. IoU, higher is better). Our method outperforms baselines at both ranking and localization. | 63 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | Levels of visual recognition. Left: assign a category to an image. Middle: describe an image with a sentence. Right: jointly detect and describe all salient aspects of the image. | 2 |
| 2.1 | Diagram of the data flow in a typical supervised learning problem approached with a neural network. The input is a dataset of pairs (x, y) of examples x and labels y . We must choose the form of three functions: 1) The function f that maps the examples x to some predicted labels \hat{y} (usually a neural network in this dissertation) using also some parameters w (sometimes also denoted θ) that we will learn. 2) The function $L(\hat{y}, y)$ that evaluates the mismatch between the prediction and the true label, and 3) the function R that evaluates the complexity of the mapping. The data loss and the regularization loss are added and the entire graph produces a single scalar value measuring how well the parameters fit our data and how “simple” the mapping is. The objective becomes to find the parameters w that minimize the final loss. | 13 |

- 2.2 An example of backpropagation along a computational graph. During forward pass x and y take on specific (numerical) values and the vector (or scalar) z is computed using some fixed function (e.g. $z = x \odot y$). Notice that we can immediately compute the Jacobian matrices $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ of this transformation using calculus, because we know what function z is computing in the forward pass. These tell us what first order influence x and y have on the value of z . The value z goes off into a computational graph and eventually at the end of the graph the total loss g (a scalar) is computed. The backward pass proceeds in the reverse order, recursively applying the chain rule to find the influence of all inputs of the graph on the final output. In particular, this computational unit finds out what $\frac{\partial g}{\partial z}$ is, telling us how z influences the final graph output. The chain rule states that to backpropagate this we should take the global gradient on z , $\frac{\partial g}{\partial z}$ and multiply it onto the local gradients for each input. For example, the global gradient for x will become $\frac{\partial g}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial g}{\partial z}$. If x, z are vectors then this is a single matrix-vector multiplication. The gradient is then recursively chained, in turn, through the functions that produced the values of x and y until the inputs are reached. In neural network applications, the inputs we are interested in are the parameters, and their gradient tells us which way they should be nudged to decrease the loss. 18
- 2.3 **Left:** A diagram of the biological inspiration behind a single neuron. Inputs x_i interact multiplicatively with the synapses w_i , the cell body accumulates the sum and then fires an output signal after the activation function. If the activation is the sigmoid non-linearity (with output range in $[0,1]$), then the output can be interpreted as the average firing rate of the neuron. **Right:** an example arrangement of neurons in a 3-layer neural network. Neurons in one layer have connections to all neurons in the previous layer but are not connected to each other. This arrangement allows us to efficiently evaluate activations of all neurons in one layer with a matrix multiplication. 20
- 2.4 Illustration of convolving a 5×5 filter (which we will eventually learn) over a $32 \times 32 \times 3$ input array with stride 1 and with no input padding. The filters are always small spatially (5 vs. 32), but always span the full depth of the input array (3). There are 28×28 unique positions for a 5×5 filter in a 32×32 input, so the convolution produces a 28×28 activation map, where each element is the result of a dot product between the filter and the input. A convolutional layer has not just one but a set of different filters (e.g. 64 of them), each applied in the same way and independently, resulting in their own activation maps. The activation maps are finally stacked together along depth to produce the output of the layer (e.g. $28 \times 28 \times 64$ array in this case). . . . 21

| | | |
|-----|--|----|
| 2.5 | An ordinary neural network (left) might take an input vector (red), transform it through some hidden layer (green), and produce an output vector (blue). In these diagrams boxes indicate vectors and arrows indicate functional dependencies. Recurrent neural networks allow us to process sequences of vectors, for example: 1) at the output, 2) at the input, or 3) both either serially or in parallel. This is facilitated by a recurrent hidden layer (green) that manipulates a set of internal variables h_t based on previous hidden state h_{t-1} and the current input using a fixed recurrence formula $h_t = f_\theta(h_{t-1}, x_t)$, where θ are parameters we can learn. | 24 |
| 2.6 | An example of using a RNN as a character-level language model. The training sequence is “hello” and the vocabulary has 4 characters: h,e,l,o. The inputs are 1-hot encodings of the characters “h,e,l,l”, and we want the RNN to predict the next character in the sequence at each time step. The RNN has a 3-dimensional hidden state (green) and there are 4 dimensions in the output vectors (blue), interpreted as the logits (i.e. scores) for the next character. The loss function and the gradient will encourage the logits of correct characters (highlighted in green) to become higher, and the other logits (in red) to become lower. | 25 |
| 3.1 | Overview of the image sentence matching model. Left: A single neural network that takes an image and a sentence and computes a (scalar-valued) matching score. The network consists of three components: an image encoder that process an image into a single vector v or a set of fragment vectors $\{v\}$, a sentence encoder that processes the sentence into a single vector s or a set of fragment vectors $\{s\}$, and a module that computes the score. Right: During training we compute pairwise scores for image-sentence pairs in a batch of training data. The loss function will encourage the true image-sentence pair scores (along the diagonal, in green) to be higher than the false image-sentence pair scores (off-diagonal, in red). | 29 |
| 3.2 | Diagram for our model. The network takes an image and a sentence and computes the image-sentence matching score S . The image is encoded using the image fragment encoding into a set of vectors $\{v\}$ - the CNN features corresponding to objects detected with an R-CNN [29] and the full image (Section 3.2.1). The sentence is encoded into fragment vectors $\{s\}$ using a bidirectional RNN (Section 3.2.2). Equation 3.6 computes the pairwise inner products between all fragments across the two modalities (the magnitudes are shown in grayscale, where white is high and black is low), and then these scores are processed with a max across columns and a sum across rows to compute the image-sentence score. The model is optimized using the loss in Equation 3.5. | 37 |

| | | |
|-----|---|----|
| 3.3 | Example alignments predicted by our model. For every test image above, we retrieve the most compatible test sentence and visualize the highest-scoring region for each word and the associated scores ($v_i^T s_t$). We hide the alignments of low-scoring words to reduce clutter. We assign each region an arbitrary color. | 40 |
| 3.4 | Left: Flickr30K test set regions with high vector magnitude, indicating a strong influence on the image-sentence score. Right: This table shows the top magnitudes of vectors ($\ s_t\ $) for words in Flickr30K. Since the magnitude of individual words in our model is also a function of their surrounding context in the sentence, we report the average magnitude. | 41 |
| 3.5 | Examples of highest scoring regions for queried snippets of text, on 5,000 images of MS COCO test set images. | 42 |
| 4.1 | Diagram of the Multimodal Recurrent Neural Network model. Each arrow indicates a functional dependence. The RNN takes a word vector (green) and the context vector from previous time steps (yellow) and defines a distribution over the next word in the sentence (blue vectors). The RNN is conditioned on the image information at the first time step (the arrow annotated with the weight matrix W_{hi} . START and END are special tokens denoting the beginning and the end of the sentence. | 45 |
| 4.2 | Example sentences generated by the Multimodal RNN for test images. | 49 |
| 5.1 | Left: An image captioning model is forced to fall back on a generic caption to describe the image in one sentence. Right: The proposed DenseCap model can both localize and describe all elements of the visual scene, including the full image as a special case. | 52 |
| 5.2 | The Dense Captioning task (bottom right) simultaneously combines the challenges of object detection (high label density, x-axis) and image captioning (high label complexity, y-axis). That is, the model must both detect and describe all salient aspects of the visual scene. Importantly, a dense captioning model can also allow the converse task of using a query description to search an image collection and identify matching regions. | 53 |
| 5.3 | Model overview. An input image is first processed a CNN. The Localization Layer proposes regions and smoothly extracts a batch of corresponding activations using bilinear interpolation. These regions are processed with a fully-connected recognition network and described with an RNN language model. The model is trained end-to-end with gradient descent. | 55 |
| 5.4 | Example captions generated and localized by our model on test images. We render the top few most confident predictions. On the bottom row we additionally contrast the amount of information our model generates compared to the Full image RNN. | 60 |

| | | |
|-----|--|----|
| 5.5 | Additional predictions from the DenseCap model shown in a different format. The captions are shown both in the boxes but also listed below, in a decreasing order of their prediction confidence. The colors are arbitrary. | 61 |
| 5.6 | Example image retrieval results using our dense captioning model. From left to right, each row shows a grund-truth test image, ground-truth region captions describing the image, and the top images retrieved by our model using the text of the captions as a query. Our model is able to correctly retrieve and localize people, animals, and parts of both natural and man-made objects. | 64 |
| 5.7 | Example results for open world detection. We use our dense captioning model to localize arbitrary pieces of text in images, and display the top detections on the test set for several queries. | 65 |
| 6.1 | An image depicting an amusing situation. | 68 |

1

Introduction

1.1 Overview

“We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc. Again I do not know what the right answer is, but I think both approaches should be tried.”

– Alan Turing, *Computing Machinery and Intelligence* (1950)

Following Alan Turing’s inspiring vision, one of the dreams of the field of Artificial Intelligence is to enable computers to see and understand the rich visual world around us and endow them with the ability to communicate with us in natural language.

Humans find it easy to accomplish a wide variety of tasks that involve complex visual recognition and scene understanding, tasks that involve communication in natural language and tasks that combine translation between the two modalities. For instance, a quick glance at an image is sufficient for a human to point out and describe an immense amount of details about the visual scene. Using the example in Figure 1.1, we can look at the image and immediately point out and describe the “orange spotted cat”, the “skateboard with red wheels”, the “brown hardwood floor”, or simply say that the entire image is a “cat riding a skateboard”.

Challenges. Since this ability feels so natural and effortless for us it can be easy to forget how difficult this task is for a computer. In a computer, this image is represented as one large array of numbers indicating the brightness at any position. An ordinary image might have a few



Figure 1.1: Levels of visual recognition. **Left:** assign a category to an image. **Middle:** describe an image with a sentence. **Right:** jointly detect and describe all salient aspects of the image.

million of these *pixels* and a computer must transform these patterns of brightness values into high-level, semantic concepts such as a “cat”. Moreover, a different breed of cat seen under different lighting conditions, with a different camera angle, or in a different pose might still depict a “cat riding a skateboard”, but the pattern of brightness values could be completely different. Conversely, patterns with very similar low-level statistics (e.g. fur-like high frequency patterns) might instead be part of many different objects (carpets, coats, etc.) or animals (dogs, bears, monkeys, etc.). The challenges are no less severe on the language side. A natural language description such as “cat riding a skateboard” will be represented in the computer as a sequence of integers indicating the index of each word in a vocabulary (e.g. “cat riding a skateboard” might be [252, 823, 18, 1742]). Therefore, the very natural task of pointing out and naming different parts of an image in fact involves a complex pattern recognition process of identifying salient subsets of a grid of a few million brightness values and annotating them with sequences of integers. Moreover, the image captions we will work with in this dissertation often require detecting and describing complex high-level concepts that are not only visual but require difficult inferences. For example, some images can be annotated by humans as “a group of men fighting”, which requires the ability to detect multiple people and analyze their poses, spatial arrangements or even their facial features. Alternatively, someone could be described as “waiting” for something, “playing” with something, or “playing a joke” on someone.

Encouraging progress. Despite the difficulty of this task, we have recently witnessed rapid progress in the area of visual recognition. In particular, the state of the art image recognition models based on deep convolutional neural networks [58] have become capable of distinguishing thousands of visual categories at accuracies comparable to humans, or even surpassing them in some fine-grained categories such as breeds of dogs [84]. Our progress on related tasks such as segmentation and object detection has been similarly dramatic [81, 107]. Together, these advances have enabled many real-world applications including face detection and recognition, personal photo search, perception in robotics and self-driving cars, etc.

Remaining challenges. However, the predominant approach in most of these applications is to model the visual recognition problem as a task of classifying images into some number of fixed and hard-coded visual categories (Figure 1.1, left). For instance, the ImageNet visual recognition challenge [84] (a popular visual recognition benchmark) consists of a set of 1,000 categories that were

picked manually by the organizers (examples include “hot dog”, “screwdriver”, “jellyfish”, “notebook”, “Yorkshire terrier”, and also perplexingly exclude many common concepts such as “person”, “face”, etc.). Similarly, the PASCAL VOC [24] object detection benchmark uses a different set of 20 manually chosen categories (e.g. “car”, “person”, “potted plant”, etc.). Other datasets for scene classification, action classification or attribute classification use their own sets of categories. While visual categories constitute a convenient modeling assumption, this approach pales in comparison to the complexity of descriptions that humans can compose for images (e.g. Figure 1.1, right).

Outline of contributions. In this dissertation we develop models and techniques for using natural language as a label space for computer vision tasks. For instance, a model should be able to look at an image and describe all of its visual content in natural language instead of merely assigning it a category. Conversely, given a natural language description the model should be able to identify visual regions that depict that description. In other words, our goal is to connect the two modalities of vision and natural language and enable computers to translate between them.

Long-term motivations. These aspirations can be motivated both on a long-term scale of building up towards intelligent machines and on a shorter horizon scale of offering valuable practical applications. First, these techniques are a step towards a future in which we can interact with computers in natural language, especially where these interactions can be properly grounded in physical environments. In addition, working towards artificially intelligent agents will require us to make available to computers large amounts of information about how our world works. Concretely, there are two massive sources of knowledge we can draw on: the physical domain that contains information about the world, scenes, objects and interactions (accessed with highest bandwidths through visual sensors) and the digital domain of the Internet that contains a vast amount of semantic information that cannot be inferred from physical domain alone (e.g. what happened in 1760), encoded primarily in natural language. Therefore, vision and language are the primary channels by which the knowledge of the world can be accessed and it is critical that we develop techniques that can relate information across the two domains instead of processing each independently. As a concrete short-term example, this might allow a computer to read on the Internet that a camel is “an even-toed ungulate within the genus *Camelus*, bearing distinctive fatty deposits known as “humps” on its back” and recognize such concept in visual data based on the description alone. In a longer-term future, the computer could read about procedures (e.g. cooking, fixing something broken, etc.) or about stories or events that unfolded in the past and understand their precise grounding in the physical world, which could unlock critical inferences not mentioned in the text alone (e.g. it’s clear what would happen if I “lift a heavy ball above my head and let go”).

Short-term motivations. The aspirations to connect the two modalities can also be motivated with more concrete, short-term and practical arguments. First, using natural language as a label space for visual recognition offers many appealing practical properties. Language is a rich encoding that can naturally represent nouns (objects, people, scenes), adjectives (attributes) verbs (actions)

and nested constructs that denote relationships. Therefore, predicting natural language utterances generalizes and inherits the challenges of many other visual recognition tasks that are currently treated as separate areas in computer vision, including object recognition, scene classification, attribute classification, action recognition, etc. Second, the end users of our Computer Vision systems are humans, who are already fluent in natural language. Therefore, if computers used language as a label space we could enable significantly easier and more natural interactions between computers and humans, without having to resort to translations between natural language and fixed categories at input or output of our systems. For example, searching a personal photo collection with arbitrary text queries such as “pictures of me swimming with friends next to a waterfall” would not have to go through intermediate stages of invoking action classifiers for “swimming”, or object classifiers for “person” or the “waterfall”, but could be directly consumed by a model and treated as a *first class citizen* representation. Conversely, a computer could directly describe a visual scene to a visually impaired person, or answer queries about the scene.

Challenges of this approach. However, this approach also poses some challenges. One common criticism is that in this setting the evaluation becomes more difficult. For instance, in image classification each image is annotated by some category so it is straight forward to compute and report an accuracy - the fraction of predictions that are correct. However, if a model annotates an image with an arbitrary sentence it can be more difficult to evaluate how correct that sentence is. In a standard setting, we might have some number of reference sentences written by humans that we can compare each prediction to, but this comparison can be difficult to specify explicitly. In our work we use the state of the art automatic evaluation methods that offer the highest correlations with human judgments and expect that additional progress can be made in further improving these methods in the future. Another criticism is that this approach couples the visual recognition task (i.e. what is in this image?) with the language modeling task (i.e. how do serialize concepts into fluent text?); it may feel natural to decouple these two tasks, study them in isolation and then compose them to form the full system later. On the other hand, addressing these tasks jointly will allow us to formulate a single model that automatically discovers all of its intermediate representations during training, without us having to explicitly specify what specific features should be extracted while processing the image to support the description task.

1.2 Related Work

Images and natural language. The work in this dissertation draws inspiration from much work that has aspired to connect the image and language modalities. Some of the earliest work focused on the problem of associating nouns with image region features based on statistical models [71, 2, 20, 41] that leverage co-occurrence statistics. Some extensions of these approaches have been developed to model not just nouns but prepositions and comparative adjectives that may be used to express

relationships between objects [33]. More recently people have started to tackle more general language constructs and full sentences. For instance, the early work of Farhadi et al. [26] present a model that maps both images and sentences to a common *meaning space*, through which they can either retrieve images based on a sentence query or vice versa. However, their model was based on a restricted processing that focused on extracting a single (object, action, scene) triple from each image and sentence. The work of Ordonez et al. [73] developed a similar approach but used more general images, a larger set of object categories and scenes, and additional image aspects including stuff, attributes, etc. Similarly, Socher and Fei-Fei [89] mapped image segments and words into a common meaning space using kernelized canonical correlation analysis on feature representations in both domains. Hodosh et al. [39] adopted a similar approach on the level of full images and sentences. Since these ranking methods are constrained to annotating images from a finite collection of sentences, many approaches have been proposed to overcome this limitation and generate descriptions. The early work of Yao et al. [104] developed an approach processed an image with a hierarchical parsing engine and then serialized to text via an intermediate process of part of speech production rules that get filled in based on the image content. Unfortunately, their approach required extensive hand-coded details about the generative process. A similar approach in spirit was used earlier by Gupta et al. [34] who used AND-OR graphs to describe sport event videos. The work of Li et al. [60] instead detected objects in the image and composed them into a sentence using pretrained n-grams, which act as large lookup tables of frequently-occurring short snippet phrases. Kuznetsova et al. [53] generate captions by retrieving and then selectively combining pieces of human-written sentences. Kulkarni et al. [52] use a approach where detections of objects, scenes, modifiers or spatial attributes are inserted into fixed sentence templates. Finally, Yang et al. [103] and Mitchell et al. [70] similarly estimate likely words based on object detections and generate descriptions by growing syntactic trees with production rules.

In this dissertation, we share the aspirations of this body of work and address both the image captioning and image ranking tasks, but develop novel techniques and algorithms in the neural networks modeling paradigm which offers superior results and multiple appealing practical properties. In addition, we build on these aspirations by introducing the dense captioning task, which requires the computer to both detect and describe all salient regions of an image. Unlike the early work in this area [71, 2, 20], our descriptions consist of arbitrary sentences and noun phrases instead of a finite collection of nouns.

Sources of data. The vast majority of modeling approaches in this area fall under the category of data-driven techniques, in which a model learns from human annotation data. It is therefore important to highlight the available datasets and carefully study their statistics.

There are a few datasets that relate the visual domain with the domain of natural language. In this dissertation we use the Flickr8K [39], Flickr30K [106] and MS COCO [62] datasets which consist of a set of images (e.g. city scenes, living rooms, parks restaurants, bathrooms, etc.), each

annotated with five descriptions written by humans on the Amazon Mechanical Turk. It is also worth pointing out the distinction between a *caption* and a *description*. A *caption* is often understood as sentences that are designed to explicitly mention non-obvious aspects of the scene (such as the exact people, locations or dates) that cannot be derived from the image alone (i.e. they add information), while the latter describes the visual content in text. In the case of image-sentence datasets, the human annotators are asked to describe the content of the image with a sentence. Therefore, these datasets contain primarily images *descriptions* since it is difficult for people unfamiliar with the precise context of a photo to appropriately *caption* the image. In this dissertation we will use the two terms (*description*, *caption*) interchangeably but predominantly refer to a *description* instead of a *caption*.

Empirically, in this data collection task people often describe the most salient aspects of the image and gravitate towards describing people and their actions and interactions with each other or the environment. A few examples of these descriptions include “a panda bear sitting under a shady tree while eating bamboo”, “a display case filled with smart phones behind glass”, “two children snuggled next to each other while sleeping”, “four men cutting into a cake with Malaysia written on it”, and “a dog resting his head on the side of the boat looking out at the water”. Notice that this unconstrained setting offers a mix of variety of concepts: some of them visual and more easily detectable (e.g. “dog”, “bear”, “cake”) and some of them more semantic and harder to infer (e.g. “children sleeping”, or “looking out”).

In addition, we will also take advantage of the Visual Genome dataset [50], which instead contains multiple descriptions per image, each of which is additionally grounded to some bounding box region in the image. In this case, during data collection the humans were asked to identify multiple rectangular regions in the image and describe their content. Interestingly, these descriptions do not only feature descriptions of the most salient concepts, as is the case with the image captioning datasets, but also include descriptions such as “silver door handle”, or “computer monitor”, which are otherwise not very salient and are generally unlikely to be mentioned in descriptions of entire images.

Modeling approach.

Most of the modeling approaches in the work discussed so far use processing pipelines that first consist of steps of feature extraction, object detection, scene or attribute classification to extract high-level concepts from the images, and then these are related to natural language utterances with statistical models. The downsides of this approach are that the model quickly devolves into complex multi-staged pipelines with preprocessing stages, intermediate computations and representations, and a large number of hyperparameters. The object, scene or attribute detectors are also often trained on distinct datasets with their own statistics and biases and the transfer to the eventual domain of interest is limited.

Recently, due to the availability of large-scale datasets and faster computation we have witnessed

rapid progress in visual recognition with the use of deep convolutional neural networks [51, 84]. These models are based on a number of techniques developed starting in the 1940s. Under the name “cybernetics”, McCulloch and Pitts [67], Rosenblatt [82] and Widrow and Hoff [100] developed early models neurons with adjustable synaptic strengths and learning rules. In the 1980s, under the name “connectionism” and “parallel distributed processing”, Rumelhart, Hinton and Williams popularized the use of backpropagation for training networks of neurons [83], which is used to this day as an exceedingly effective credit assignment algorithm based on efficiently evaluating (by a recursive application of the chain rule) the gradient of a loss function with respect to the parameters of the network. In particular, one of the first successful applications of these techniques in visual recognition was the work of LeCun et al. [57] for digit recognition, which similar to Fukushima’s Neocognitron [28] arranged neurons in a lattice structure and a local connectivity in space. This was the first description of a modern convolutional neural network, but its success on large-scale visual recognition problems (such the ImageNet challenge [84]) was only possible a few decades later with significantly more computation and training data [51] (under a new name of “deep learning”). Convolutional networks describe a function from an input space (e.g. images) to an output space (e.g. probabilities for some number of classes) and the parameters of this function (the synaptic strengths of the connections of all neurons) are trained using a large collection of labeled images (e.g. ImageNet dataset). We say that in this approach the model is trained “end-to-end”, meaning that the entire computational process from the inputs to the outputs shares the same end objective (correctly classifying images) and is jointly optimized over. This property is widely recognized as one of the core properties that makes these models work so well in practice. Another practical advantage of these models is that once they are trained on one dataset they can be used a fixed feature extractor for images, but more importantly they can be “finetuned” on a different dataset by initializing the parameters from the first task and continuing to train them on a second [17, 87], which often leads to sizable improvements in the final performance. In our work we use convolutional neural networks as a core processing module for images, but finetune them as part of larger neural network architectures that relate the images to language constructs.

In addition to progress in visual recognition, we have witnessed similar successes in natural language processing. In particular related to this dissertation, Mikolov et al. [69] have demonstrated that it is possible to learn distribution representations [37] of words such that words that occur in similar contexts are found nearby in the vector space. For example, the words “woman” and “girl” might (after training on a large collection of sentences) be found nearby each other, indicating a similar semantic meaning. Similar to convolutional networks, we can repurpose these word embeddings in other applications and further “finetune” them on new tasks. Lastly, we take advantage of work on neural language models [4], that formulate neural network architectures that represent probability distributions over sequences of words (i.e. sentences). More recently, recurrent neural networks [99, 38] have proven especially effective in the language modeling task [68, 92, 31]. In our

work we use language models and techniques in larger architectures that represent the conditional distribution over sentences given an image.

In summary, in this dissertation we adopt the end-to-end learning paradigm and design neural network architectures for the tasks of image-sentence matching, image captioning tasks and image region-annotation. As a result, our models are more computationally homogenous, simpler and yield superior results due to the benefits of end-to-end training and transfer learning from large-scale related datasets such as ImageNet [14].

1.3 Contributions and Outline

In this dissertation we develop models for connecting images with natural language. In particular, we develop neural network architectures that process and align the two modalities and train their parameters end-to-end on datasets of image captions.

In **Chapter 2** we provide relevant mathematical background for supervised learning, backpropagation, optimization, neural networks, and describe commonly used architectural design patterns for processing images and text, especially convolutional and recurrent neural networks.

In **Chapter 3** we develop a model that can match images and sentences. That is, given a finite set of images and sentences, we can pick an image and rank the sentences based on how compatible they are with that image (how well they describe its content) and conversely, we can pick a sentence and rank the images based on how compatible they are with that sentence (how well they depict that description). The model will process the image and the sentence independently and embed them into a common multimodal embedding space, in which inner products correspond to intermodal similarities. The content of these chapters is based primarily on the ranking model section of Karpathy and Fei-Fei [43], and to a much lesser degree on earlier work from Karpathy et al. [45] and Socher et al. [90].

In **Chapter 4** we specifically address the problem of generating novel descriptions for images and relax the restriction of the ranking model that assumes a finite collection of sentences to choose from. The model can therefore take an image as input and generate a novel description that might not appear verbatim in the training data. The architecture is based on a combination of a convolutional network for processing the image and a recurrent neural network language conditioned on the image information. The content of this chapter is based on the generation model section of Karpathy and Fei-Fei [43] and to a much lesser degree on Karpathy, Johnson and Fei-Fei [44], where Karpathy and Johnson are equal co-authors.

In **Chapter 5** we remove the restriction that an image must be described with a single sentence and present a model that can be both detect and describe all salient parts of an image. Therefore, this model may choose to describe the entire image as seen in the previous chapter, but in addition it can also spatially localize and describe many other parts of an image, such as people, backgrounds,

cell phones, etc. The architecture of this model comprises a convolutional neural network followed by a region proposal module inspired by object detection methods and a recurrent neural network language model for describing all detected regions of interest. The content of this chapter are based on the work of Johnson, Karpathy and Fei-Fei [42] where Johnson and Karpathy are equal first co-authors.

Finally, in **Chapter 6** we identify the remaining challenges and discuss the path forward.

2

Deep Learning Background

This chapter provides the necessary technical background on machine learning and neural networks. For a more thorough and slower-paced introduction we recommend the Deep Learning book from Goodfellow et al. [3].

2.1 Supervised Learning

Many practical problems can be formulated as requiring a computer to perform a mapping $f : X \rightarrow Y$, where X is an input space and Y is an output space. For instance, in visual recognition X could be the space of images and Y could be the interval $[0, 1]$ indicating the probability of a cat appearing somewhere in the image. Unfortunately, in many cases it is difficult to manually specify the function f by conventional means (e.g. it is unclear how one might write down a program that recognizes a cat). The supervised learning paradigm offers an alternative approach that takes advantage of the fact that it is often relatively easy to obtain examples $(x, y) \in X \times Y$ of the desired mapping. In our running example, this would correspond to collecting a dataset of images each labeled with the presence or absence of a cat, as annotated by humans.

The objective. Concretely, we assume a training dataset of n examples $\{(x_1, y_1), \dots, (x_n, y_n)\}$ made up of independent and identically distributed (i.i.d.) samples from a data generating distribution D ; i.e. $(x_i, y_i) \sim D$ for all i . We then think about *learning* the mapping $f : X \mapsto Y$ by searching over a set of candidate functions and finding the one that is most consistent with the training examples. More precisely, we consider some particular class of functions \mathcal{F} and choose a scalar-valued loss function $L(\hat{y}, y)$ that measures the disagreement between a predicted label $\hat{y}_i = f(x_i)$ for some $f \in \mathcal{F}$ and a true label y_i . Our objective in learning is to find $f^* \in \mathcal{F}$ that ideally satisfies:

$$f^* = \arg \min_{f \in \mathcal{F}} E_{(x,y) \sim D} L(f(x), y). \quad (2.1)$$

In other words, we seek a function f^* that minimizes the expected loss over the data generating distribution D . In practical applications, once we identify this function we can discard the original training data and only keep the learned function f^* , which we use to map elements of X to Y .

Unfortunately, the optimization problem above is intractable because we do not have access to all possible elements of D and therefore cannot evaluate the expectation or simplify it analytically without making unrealistically strong assumptions about the form of D, L or f . However, under the i.i.d. assumption we can approximate the expected loss in Equation 2.1 above with sampling by averaging the loss over the available training data:

$$f^* \approx \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(x_i), y_i). \quad (2.2)$$

In other words we optimize the loss only over the available training examples, but the hope is that this is a good proxy objective for the actual objective in Equation 2.1.

Regularization. Unfortunately, optimizing Equation 2.2 instead of Equation 2.1 poses challenges. For instance, consider a function f that maps each x_i in the training data to its y_i but returns zero everywhere else. This would be a solution to Equation 2.2 (for any sensible loss function L that achieves a minimum value when $y = \hat{y}$), but we would expect very high loss for all other points in D that are not in the training set. In other words, we would not expect this function to *generalize* to all $(x, y) \sim D$. An additional less esoteric concern is that there may be many different functions that all achieve the same loss under Equation 2.2 (so there is no unique solution), but their generalization outside of the training data could vary. If all we have are the training data then how do we choose among an entire set of $f \in \mathcal{F}$ that all achieve the same loss in Equation 2.2? Both of these concerns can be alleviated by introducing a regularization term R to the objective:

$$f^* = \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(x_i), y_i) + R(f), \quad (2.3)$$

where R is a scalar-valued function that encodes preference for some functions over others, regardless of their fit to the training data. This addition can be partly justified as following the principle of Occam's razor, which could be stated as: "*Suppose there exist two explanations for an occurrence. In this case the simpler one is usually better*". Put in another way, the regularization is a measure of complexity of a function. Together with the regularization term, the objective in Equation 2.3 encourages simple solutions that also fit the training data well, and its intended effect is to some extent compensate for the discrepancy between the objective in Equation 2.2 and Equation 2.1.

Example: Linear regression. Consider a simple example where we are given a dataset of 100 ($n = 100$) 2-dimensional points ($X = \mathbb{R}^2$) each annotated with a scalar ($Y = \mathbb{R}$). The hypothesis class \mathcal{F} we may consider is the set of linear functions from X to Y (i.e. $\mathcal{F} = \{w^T x + b \mid w \in \mathbb{R}^2, b \in \mathbb{R}\}$). In this case our hypothesis space is spanned by 3 parameters (w_1, w_2, b) , where we $w = [w_1, w_2]$.

A commonly used loss function in a regression setting is the (squared) difference between the target and the predicted value, $L(\hat{y}, y) = (\hat{y} - y)^2$. Finally, as a commonly used regularization we could use $R(w, b) = \lambda(w_1^2 + w_2^2)$ which discourages the parameters w_1, w_2 from being too large and causing one input feature to have a disproportionate effect on the predictions, and λ is also a parameter specifying the strength of the regularization. Putting these pieces together, the problem is to solve:

$$f^* = \arg \min_{w, b} \left[\underbrace{\frac{1}{n} \sum_{i=1}^n (w^T x_i + b - y_i)^2}_{\text{fit the training data}} \right] + \left[\underbrace{\lambda(w_1^2 + w_2^2)}_{\text{regularization}} \right].$$

In particular also note that it is common to exclude the bias terms from the regularization since these do not interact multiplicatively with inputs; they merely allow the model to offset itself away from the origin.

Example: Neural network regression. As a brief preview, to extend the above example to a neural network it suffices to make the hypothesis space \mathcal{F} more complex. For instance, instead of searching over linear functions of form $f(x) = w^T x + b$ we could use $f(x) = w_2 \tanh(W_1^T x + b_1) + b_2$, where W_1, w_2, b_1, b_2 are all parameters: W_1 is a matrix of size $H \times 2$, b_1 is a vector of size H , w_2 is a vector of size H , and b_2 is a scalar. Here H is an integer that we are free to choose (e.g. 100; it is often interpreted as the number of neurons in the hidden layer) and the hyperbolic tangent \tanh is applied elementwise (it squashes values to the interval $[-1, 1]$ and is in neural networks context commonly referred to as a *non-linearity*). The objective thus becomes:

$$f^* = \arg \min_{W_1, b_1, w_2, b_2} \left[\underbrace{\frac{1}{n} \sum_{i=1}^n (w_2 \tanh(W_1^T x_i + b_1) + b_2 - y_i)^2}_{\text{fit the training data}} \right] + \left[\underbrace{\lambda(\|W_1\|_2^2 + \|w_2\|_2^2)}_{\text{regularization}} \right].$$

Example: Neural network classification. Instead of predicting some scalar-valued quantity for each input, a common practical setting is that of classification where one wants to assign the input a discrete category. For instance, using three possible classes we could use $f(x) = W_2 \tanh(W_1^T x + b_1) + b_2$, where W_2 is now a $K \times H$ matrix ($K = 3$ in this example), so the output of f is a 3-dimensional vector. It is common to interpret the numbers in this vector as logits, and hence compute the probabilities of the three classes by passing this vector through the softmax function, which takes a vector z and outputs a vector of the same size p , where $p_i = e^{z_i} / \sum_{k=1}^K e^{z_k}$. Note that the vector z can contain arbitrary real-valued quantities, but the vector p is normalized so that all of its elements are between 0 and 1 and they sum to 1 (e.g. $[0.2, 0.5, 0.3]$). This vector p is essentially our prediction \hat{y} from the network and the *correct* label y is a 3-dimensional vector that is all zero except for a single 1 at the index of the true class. For example, if the correct class is the third one then $y = [0, 0, 1]$. The most commonly used loss function in the classification setting is

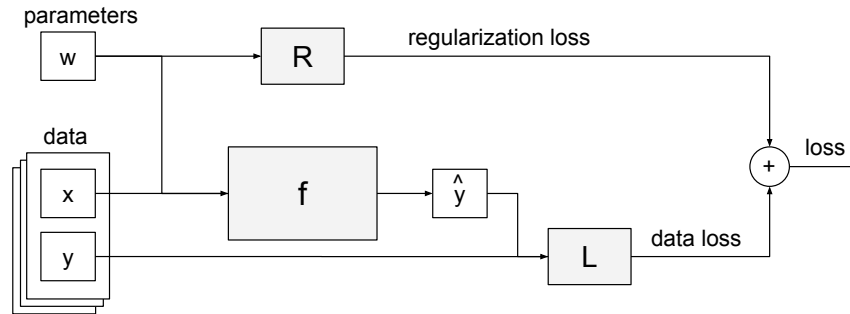


Figure 2.1: Diagram of the data flow in a typical supervised learning problem approached with a neural network. The input is a dataset of pairs (x, y) of examples x and labels y . We must choose the form of three functions: 1) The function f that maps the examples x to some predicted labels \hat{y} (usually a neural network in this dissertation) using also some parameters w (sometimes also denoted θ) that we will learn. 2) The function $L(\hat{y}, y)$ that evaluates the mismatch between the prediction and the true label, and 3) the function R that evaluates the complexity of the mapping. The data loss and the regularization loss are added and the entire graph produces a single scalar value measuring how well the parameters fit our data and how “simple” the mapping is. The objective becomes to find the parameters w that minimize the final loss.

the cross-entropy loss, which has the form:

$$L(\hat{y}, y) = - \sum_{k=1}^K y_k \log \hat{y}_k = - \log \hat{y}_{y=1},$$

where the first equality is the definition of the cross-entropy between two distributions $H(p, q) = - \sum_x p(x) \log q(x)$, and the second equality simplifies the expression because the true distribution in classification settings are usually assumed to have all of their probability mass on the single correct element, whose integer index we denote $y = 1$. Since we interpret the output of the network as containing probabilities of the three different classes, we also see that we are effectively minimizing the negative log probability of the correct class, consistent with a probabilistic interpretation of this loss as maximizing the log likelihood of the class y conditioned on the input x .

Summary. In supervised learning we are given a dataset of n datapoints $\{(x_1, y_1), \dots, (x_n, y_n)\}$ where $(x_i, y_i) \in X \times Y$ and we identify three quantities to formalize the problem:

1. The search space of functions \mathcal{F} , where each $f \in \mathcal{F}$ maps X to Y .
2. The scalar-valued loss function $L(\hat{y}, y)$ that evaluates the mismatch between a true label y and a predicted label $\hat{y} = f(x)$.
3. The scalar-valued regularization loss $R(f)$ that measures the complexity of a mapping.

Most commonly in deep learning the space of functions \mathcal{F} will be a neural network with some

parameters as seen in the example above, the loss L will be a euclidean loss in regression or a cross-entropy loss in classification, and the regularization R is most commonly the L2 norm (i.e. sum of squares of all weights).

Once these choices are made, the problem of learning a model for a supervised learning task reduces to an optimization problem of the general form $\theta^* = \arg \min_{\theta} g(\theta)$, where θ is a parameter vector and $g(\theta) = \frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i), y_i) + R(f_{\theta})$. Here we are making it clear that the parameters θ usually belong entirely to the function mapping f , and the functions L, R do not involve any parameters. We now turn to the process of solving this optimization problem in practice.

2.2 Optimization

In the last section we saw that we can reduce the task of learning a model for a supervised learning problem to solving an optimization problem of the form $\theta^* = \arg \min_{\theta} g(\theta)$, where θ is a parameter vector and g usually combines the average loss of all examples and a regularization penalty.

Derivative free optimization. First, observe that we can evaluate $g(\theta)$ for any arbitrary θ so one approach to solving this optimization problem is to use stochastic hill-climbing methods that effectively “guess-and-check”. For instance, one can draw a large number of θ at random from some distribution, check each one, and take the one that minimizes g . A more elaborate approach might iteratively seek to improve some candidate θ by repeatedly making small perturbations. Unfortunately, typical neural networks we want to train might have parameter vectors with several million or billion parameters, so many of these approaches are computationally intractable.

First order methods. We can improve the efficiency of the optimization by making additional assumptions about g . In particular, if we restrict ourselves to only using differentiable functions then we can compute the gradient $\nabla_{\theta} g$ with backpropagation (the details of this process will be discussed in the next section). The gradient is a vector of partial derivatives, giving us the slope of g along every dimension of θ . The gradient allows us to construct the first order approximation in the Taylor expansion of g , hence the name “first order methods”. We can use the gradient as a search direction; in particular, we can improve θ (in the sense of achieving lower g) by adding to it a small amount of the negative gradient direction (since we want to minimize g , but the gradient gives the direction of increase of g). This insight motivates the gradient descent (GD) algorithm that alternates the two steps: 1) evaluate the gradient with backpropagation and 2) update the parameters by taking a small step in the direction of the negative gradient. As a last practical consideration, the datasets we use in practice can be very large (e.g. ImageNet has 1 million training images) so we only estimate the gradient using a small minibatch of examples (e.g. around 100) at a time. This allows us to perform many approximate updates instead of fewer exact updates - a strategy that works well in most practical applications. The resulting algorithm, Stochastic Gradient Descent (SGD) is summarized in Algorithm 1.

Algorithm 1 Stochastic Gradient descent.

Given a starting point $\theta \in \text{dom}g$ **Given** a step size $\epsilon \in R^+$ **repeat**

1. Sample a minibatch of m examples $\{(x_1, y_1), \dots, (x_m, y_m)\}$ from training data
2. Estimate the gradient $\nabla_{\theta}g(\theta) \approx \nabla_{\theta} \left[\frac{1}{m} \sum_{i=1}^m L(f_{\theta}(x_i), y_i) + R(f_{\theta}) \right]$ with backpropagation
3. Compute the update direction: $\Delta\theta := -\epsilon \nabla_{\theta}g(\theta)$
4. Perform a parameter update: $\theta := \theta + \Delta\theta$

until convergence.

A critical parameter in SGD is the step size ϵ (also called the *learning rate*). If it is too high the optimization may not converge or even diverge. If it is set too low learning will take too long. One toy example to consider minimizing the function $y = x^2$ (which has the gradient $\frac{dy}{dx} = 2x$) with gradient descent starting from $x = 1$. Then $\epsilon > 1$ will cause gradient descent to diverge to infinity, $\epsilon = 1$ will cause the optimization to oscillate indefinitely between $x = 1$ and $x = -1$, and $\epsilon < 1$ will lead to convergence to the minimum at $x = 0$. In particular, the optimal learning rate in this toy example is just at the edge of divergence, $x = 0.9999$. In practice a good heuristic is to binary search by hand to find the lowest setting of ϵ that makes the optimization diverge (e.g. 0.1), and set the initial learning rate to be slightly smaller than this amount (e.g. 0.05) as a safety margin. It is a good idea to anneal this learning rate during training and a good rule of thumb is to reduce it by a factor of approximately 100 by the end of the optimization. This can either be done by multiplying by a constant (e.g. 0.1 twice) at fixed intervals, or by using an annealing schedule (e.g. $\epsilon_t = \alpha e^{-t\beta}$ for some α, β). Minor but consistent improvements can also be achieved with Polyak averaging [78], where we compute an averaged parameter vector $\bar{\theta}$ (e.g. $\bar{\theta} = 0.999\bar{\theta} + 0.001\theta$) after every parameter update and using $\bar{\theta}$ at test time. In general, different settings will work better or worse for different problems, so it is common to determine problem-specific settings using cross-validation, which we will discuss later in this section.

Advanced first order optimization techniques. In practice one can often obtain faster converge by modifying the computation of the update direction (Step 3 in Algorithm 1). For instance, one of these techniques is the **Momentum** update, designed to encourage progress along small but consistent directions of the gradient. Using a shorthand notation of $\mathbf{g} = \nabla_{\theta}g(\theta)$ for the gradient vector, the update $\Delta\theta$ is computed first by updating an intermediate variable $\mathbf{v} := \alpha\mathbf{v} + \mathbf{g}$ (initialized at zero), and then computing the update as $\Delta\theta := -\epsilon\mathbf{v}$. notice that the variable \mathbf{v} contains an exponentially-decaying sum of previous gradient directions. This update has a close relationship to physics, where the gradient is interpreted as a force \mathbf{F} on a particle with position θ , incrementing its velocity instead of its position directly. This is consistent with Newton's second law $\mathbf{F} = \frac{d\mathbf{p}}{dt}$, where momentum $\mathbf{p} = m\mathbf{v}$, and the mass m is assumed to be a constant.

Momentum modulates the update with a running estimate of the first moment of the gradient (its mean). A number of methods have also been developed that modulate the update using the second

moment. For example, the **Adagrad** update [19] uses an intermediate variable $\mathbf{r} := \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ of sum of squared gradients (\odot is elementwise multiplication). The second moment then modulates the update as follows: $\Delta\theta := -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$, where δ is a small number (e.g. $1e^{-5}$), preventing division by zero. The **RMSProp** update [96] instead uses a running mean of the second moment: $\mathbf{r} := \rho\mathbf{r} + (1 - \rho)\mathbf{g} \odot \mathbf{g}$, where ρ is usually set to 0.99 or so. Finally, the **Adam** update [46] estimates both first and second running moments and can be seen as a combination of RMSProp with Momentum. These methods have an equalizing effect on parameter updates: parameters that see large gradients will take smaller steps and parameters that see very low gradients will take larger steps.

Cross-validation. We saw that the problem formulation and the optimization require many settings that we left unspecified - for example the regularization strength λ , the step size for stochastic gradient descent ϵ , the number of hidden units in a neural network H , etc. Many of these parameters are difficult to attach to the parameter vector θ and train with gradient descent. Instead, we resort to stochastic optimization techniques; that is, we try several possibilities, optimize the model in each case and finally evaluate the predictions on a withheld, *validation set* of examples that were not used during training. This is a way of estimating the generalization error. In cases where not too many examples are available it is also possible to split the dataset into some number of *folds*, use one fold for validation and the rest for training, cycle over all possible choices of the validation fold, and report the average validation performance across all folds. This process is referred to as *cross-validation*, or *k-fold cross-validation* (e.g. 10-fold cross-validation in case of 10 folds). However, it is common to hear people say “cross-validation” even when they only use a single validation fold.

2.3 Backpropagation

We saw that if we can evaluate the gradient of the loss function then we can use stochastic gradient descent to minimize it, and in the process find mappings $f \in \mathcal{F}$ that achieve low regularization cost and map X to Y consistent with the training data.

We now discuss backpropagation - the process by which we efficiently compute gradients of scalar valued functions with respect to their inputs. The backpropagation algorithm is a recursive application of the chain rule from calculus. Recall that the function we are interested in computing gradients of is g , which takes as input the dataset of examples (x_i, y_i) and the parameters θ . We are specifically interested in the gradient $\nabla_{\theta} g$ with respect to the parameters θ in order to perform the parameter update but we could, if we wanted to, also compute the gradients for the inputs x_i with the same process.

Toy example. Using an example, suppose that we had an expression $y = (2x + 3)^2$ and that we are interested in computing $\partial y / \partial x$. In a neural network application x would be a vector containing both the input data and the network parameters and y would be the total loss. Introducing intermediate variables for clarity, we have that the output y is a function of the input x through a sequence

of a few intermediate functions $a = 2x$, $b = a + 3$, and $y = b^2$. In addition, note that the gradient of every individual transformation with respect to its input is easy to write down: $\partial a/\partial x = 2$, $\partial b/\partial a = 1$, and $\partial y/\partial b = 2b$. Once we know these (local) derivatives, we can obtain the desired (global) derivative $\partial y/\partial x$ by repeatedly applying the chain rule, which tells us that: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial x}$.

General statement. Looking at the example above and generalizing, we would like to multiply the individual local derivatives of all intermediate functions together to obtain the final derivative of the output with respect to the input. Using more general notation, suppose we had an input vector x_0 that we transform through a series of functions $x_i = f_i(x_{i-1})$ where $i = 1, \dots, k$ and the last x_k is a scalar. We assume that the gradient (or at least the subgradient) exists, so we can calculate the Jacobian matrix $\frac{\partial x_i}{\partial x_{i-1}}$ of all intermediate transformations, which tells us how every output dimension of x_i depends on every input dimension of x_{i-1} . By chain rule the final gradient we are interested in is, analogous to our example, simply the matrix product of all the Jacobians:

$$\frac{\partial x_k}{\partial x_0} = \prod_{i=1}^k \frac{\partial x_i}{\partial x_{i-1}}.$$

Practical considerations. It's important to note that in most neural network applications x_0 is large (e.g. all pixels of an image) while x_k is small (usually one, since we have a scalar-valued loss function). Therefore, it becomes computationally important to evaluate the above product of all Jacobians from end to front - i.e. starting with $\frac{\partial x_k}{\partial x_{k-1}}$ down to $\frac{\partial x_1}{\partial x_0}$, instead of the other way around. It's best to see this with a concrete example; if x had 10 dimensions then in our example above the product $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial x}$ would require multiplying three matrices of sizes: $[1 \times 10] \times [10 \times 10] \times [10 \times 10]$. Going from front (right) to back (left) requires $[10 \times 10] \times [10 \times 10]$ matrix multiplication followed by $[1 \times 10] \times [10 \times 10]$, while going from back to front merely requires $[1 \times 10] \times [10 \times 10]$ twice. More generally, if there are many inputs and a single output (as is the case in most neural network applications) it is always more computationally efficient to go back to front (this is called *reverse-mode differentiation*). Conversely, if there was a single input and many outputs then it would be more efficient to go front to back (*forward-mode differentiation*). Unfortunately, reverse mode differentiation requires us to keep all intermediate values computed during the forward pass in memory because we need them as we backpropagate the gradients backward later - this is a price in memory capacity we pay for computational efficiency.

Note that in principle for arbitrary functions the Jacobian matrices $\frac{\partial x_i}{\partial x_{i-1}}$ could be very large. However, in most practical applications these matrices have very special structure and we do not have to explicitly create them in memory. For example, an elementwise non-linearity \tanh that takes a 4096-dimensional vector and applies \tanh to all dimensions has a Jacobian matrix of size 4096×4096 . However, since the transformation acts elementwise, the Jacobian matrix is almost entirely zero except for elements along the diagonal. It would be wasteful to explicitly create this matrix in memory and perform a full matrix multiplication; instead, since the Jacobian only has elements along the identity for this specific operation, the matrix multiplication can be implemented very efficiently by only doing an elementwise multiplication with the elements along the diagonal.

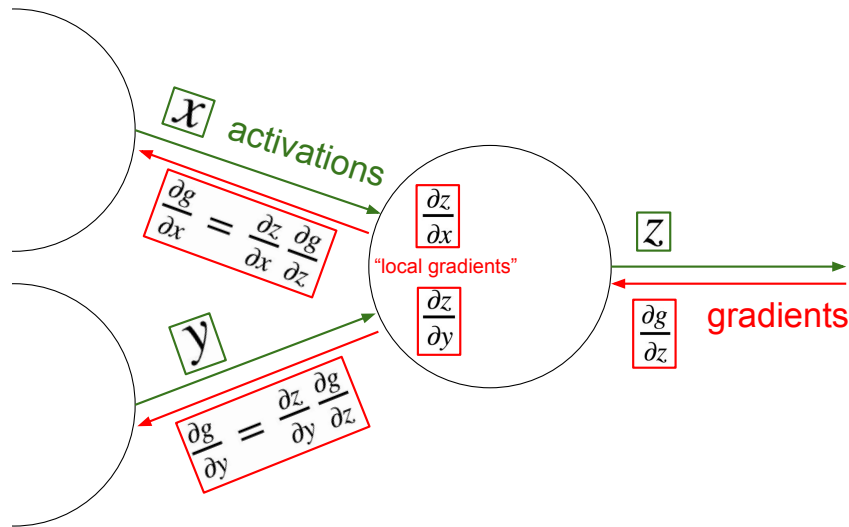


Figure 2.2: An example of backpropagation along a computational graph. During forward pass x and y take on specific (numerical) values and the vector (or scalar) z is computed using some fixed function (e.g. $z = x \odot y$). Notice that we can immediately compute the Jacobian matrices $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ of this transformation using calculus, because we know what function z is computing in the forward pass. These tell us what first order influence x and y have on the value of z . The value z goes off into a computational graph and eventually at the end of the graph the total loss g (a scalar) is computed. The backward pass proceeds in the reverse order, recursively applying the chain rule to find the influence of all inputs of the graph on the final output. In particular, this computational unit finds out what $\frac{\partial g}{\partial z}$ is, telling us how z influences the final graph output. The chain rule states that to backpropagate this we should take the global gradient on z , $\frac{\partial g}{\partial z}$ and multiply it onto the local gradients for each input. For example, the global gradient for x will become $\frac{\partial g}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial g}{\partial z}$. If x, z are vectors then this is a single matrix-vector multiplication. The gradient is then recursively chained, in turn, through the functions that produced the values of x and y until the inputs are reached. In neural network applications, the inputs we are interested in are the parameters, and their gradient tells us which way they should be nudged to decrease the loss.

In summary, evaluating the gradient of the output with respect to the input reduces to, by the chain rule, a product of Jacobian matrices. In neural network applications we first perform a *forward pass* in which we take a batch of data $\{(x_i, y_i)\}_{i=1}^m$ and current parameters θ and “forward” the network to compute all intermediate values (caching them for later) and the cost g . Then during backpropagation we proceed backwards through all intermediate stages (the “backwards pass”) and “chain” the local gradients, each time matrix multiplying by the next Jacobian matrix in the full product. In many cases we do not need to actually create the full Jacobian matrices and can take advantage of special structure to chain the gradients in a more computationally efficient manner.

Computational Graph. Instead of thinking of the computational process of a Neural Network as a linear list of operations it is more intuitive to think about the function from inputs to outputs

as a directed acyclic graph (DAG) of operations, where vectors flow along edges and nodes represent differentiable transformations that consume some number of vectors and combine them to one vector that then flows to other nodes. Most implementations of backpropagation organize the code base around a *Graph* object that maintains the connectivity of operations (also called *gates*, or *layers*) and a large collection of possible operations. Both *Graph* and *Node* objects implement two functions: `forward()` and `backward()`. The *Graph*'s `forward()` iterates over all nodes in the topological order and calls their `forward()`, and its `backward()` iterates in the reverse order and calls their `backward()`. Each *Node* computes its output with some function during its `forward()` call, and in `backward()` it is given the gradient of the loss function with respect to its output and returns the “chained” gradients on all of its inputs. By “chained” we mean taking the gradient on its output and multiplying it by its own local gradient (the Jacobian of this transformation). This gradient then flows to its children nodes that perform the same operation recursively. As a last technical note, if the output of a node is used in multiple places in the graph then correct behavior by the multivariable chain rule is to add the gradients during backpropagation along all branches. This would be handled in the *Graph* object.

As an example, a \tanh layer would implement the forward function as $y = \tanh x$, where \tanh is applied elementwise. During backpropagation the backward function will be called by the *Graph* object, supplying it with $\frac{\partial g}{\partial y}$, the “Jacobian product so far”, where g is the loss at the end of the graph. The objective of the backward function is to chain the Jacobian of this layer onto the running product, in this case computing the matrix product $\frac{\partial g}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial g}{\partial y}$. However, since the Jacobian $\frac{\partial y}{\partial x}$ of this layer only has elements along the diagonal, this matrix multiply can instead be implemented efficiently as $\frac{\partial g}{\partial x} = \mathbf{eye}(\frac{\partial y}{\partial x}) \odot \frac{\partial g}{\partial y}$, where \odot is an elementwise multiplication and $\mathbf{eye}(\frac{\partial y}{\partial x})$ is the diagonal of the matrix as a vector (for \tanh this would be the vector $(1 - y \odot y)$ since $\frac{d}{dx} \tanh x = (1 - (\tanh x)^2)$). The *Graph* object will then take the returned $\frac{\partial g}{\partial x}$ from this layer, and pass it on to the node that computed x , continuing the recursive process all the way to the inputs of the graph (the data and the parameters).

2.4 Neural Networks

In the previous sections we saw that we can define arbitrary differentiable functions f that transform the inputs x to predicted outputs \hat{y} , and optimize the model with respect to any differentiable loss function using stochastic gradient descent. We now turn to the details of the function f , which we have so far left unspecified.

2.4.1 Vanilla Neural Networks

In absence of any assumptions about the structure of x , we construct neural networks by repeating matrix multiplications and element-wise non-linearities. As an example, a 2-layer neural network

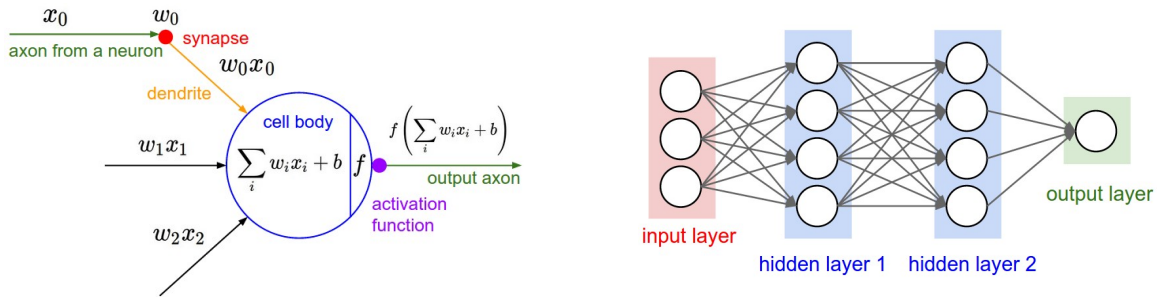


Figure 2.3: **Left:** A diagram of the biological inspiration behind a single neuron. Inputs x_i interact multiplicatively with the synapses w_i , the cell body accumulates the sum and then fires an output signal after the activation function. If the activation is the sigmoid non-linearity (with output range in $[0,1]$), then the output can be interpreted as the average firing rate of the neuron. **Right:** an example arrangement of neurons in a 3-layer neural network. Neurons in one layer have connections to all neurons in the previous layer but are not connected to each other. This arrangement allows us to efficiently evaluate activations of all neurons in one layer with a matrix multiplication.

would be implemented as $f(x) = W_2\sigma(W_1x)$, where W_1, W_2 are matrices and σ is an element-wise non-linearity (e.g. \tanh). A 3-layer network would have the form $f(x) = W_3\sigma(W_2\sigma(W_1x))$, etc. Note that if the non-linearity is an identity function then the entire neural network reduces (in representational power) to a linear function. Common settings for the non-linearities are \tanh , the sigmoid function $1/(1+e^{-x})$ and the rectified linear unit (ReLU) $\max(0, x)$. Note that the last layer of the neural network normally does not contain the non-linearity. Also note that a 1-layer neural network is a simple linear transformation.

Biological inspiration. The biological inspiration for neural networks has historically emerged from a crude model of a biological neuron. In particular, each row of the weight matrices W models one neuron and its synaptic connection strengths to its input - positive weights are excitatory, negative weights inhibitory, and a weight of zero represents no dependence. The weighted sum of the inputs (after the inner product) arrives at the cell body. The original model used a sigmoid activation function which squashes the weighted sum to the range between $[0,1]$, and is interpreted as the firing rate of the neuron. In this interpretation, each matrix multiplication by a weight matrix W and a subsequent sigmoid non-linearity evaluates, in parallel, the firing rates of several neurons arranged in a layer, where none of the neurons connect to each other and each one is connected to the same inputs in the layer before.

2.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs, or ConvNets) [58] are neural network architectures specifically designed for handling data with some spatial topology (e.g. images, videos, sound spectrograms in speech processing, character sequences in text, or 3D voxel data). In each of these cases an input example x is a multi-dimensional array (i.e. a *tensor*). E.g. a 256x256 color image is

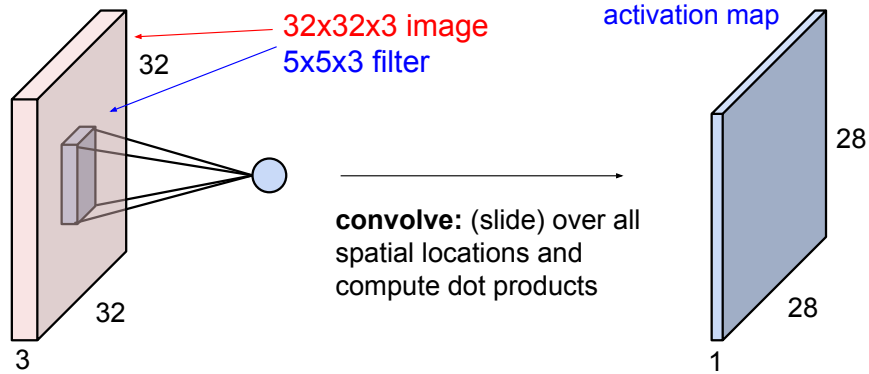


Figure 2.4: Illustration of convolving a 5×5 filter (which we will eventually learn) over a $32 \times 32 \times 3$ input array with stride 1 and with no input padding. The filters are always small spatially (5 vs. 32), but always span the full depth of the input array (3). There are 28×28 unique positions for a 5×5 filter in a 32×32 input, so the convolution produces a 28×28 activation map, where each element is the result of a dot product between the filter and the input. A convolutional layer has not just one but a set of different filters (e.g. 64 of them), each applied in the same way and independently, resulting in their own activation maps. The activation maps are finally stacked together along depth to produce the output of the layer (e.g. $28 \times 28 \times 64$ array in this case).

a $256 \times 256 \times 3$ tensor (for 3 color channels red, green, blue). A sound spectrogram could be an array of size $1,000 \times 128$, indicating the amplitude of any one of 128 frequencies at any point in time from $t = 1, \dots, 1000$. Some sentence could be represented on a character level as a 112×30 , indicating which of 30 possible characters occupies any one of 112 positions in the sentence. In many of these cases the input dimensionality is high (e.g. the image above will have approximately 200,000 numbers) and it is wasteful (in both number of parameters and processing time) to use fully connected layers as we saw in simple neural networks in the preceding section. In these cases we prefer to design neural network architectures that are aware of the spatial layout of the input and use specific local connectivity and sensible parameter sharing schemes.

Concrete example. The core computational building block of a Convolutional Neural Network is the *Convolutional Layer* (or the CONV layer) which takes an input tensor and produces an output tensor by convolving the input with a set of filters. We'll introduce the concept with a concrete example and then discuss the more general case. In this dissertation we are primarily concerned with processing images. Suppose that our input is a color image of with and height of 227 - i.e. a $227 \times 227 \times 3$ tensor X . Now consider a $5 \times 5 \times 3$ filter w , which is a tensor of $5 * 5 * 3 = 75$ parameters that we will eventually want to learn. We can *convolve* this filter by sliding it across all spatial positions of the input tensor and computing a dot product between a small chunk of X and the filter w at each position. The result will be an *activation map*, which in this case would have the dimensions 223×223 (223 is the number of unique positions that a filter of 5 elements can be placed over an input of size 227). It is common to also pad the input with a border of zeros to

control the output dimension. For instance, padding with a border of zeros of thickness of 2 in this case would result in a 227×227 output activation map. It is also possible to apply filters with some stride. For example applying the $5 \times 5 \times 3$ filter to the $227 \times 227 \times 3$ input tensor with no padding and stride of 2 would instead give an output activation map of size 112×112 (we will shortly see how we can compute these output size in general). Finally, the CONV layer does not possess only a single filter but an entire set of them. For example, if we use 32 filters then each would compute its own 112×112 activation map and these are stacked to produce the final output tensor (e.g. of size $112 \times 112 \times 32$). Intuitively, each filter has the capacity to “look for” certain local features in the input tensor and the parameters that make up the filters are trained with backpropagation.

General definition. More generally, a convolutional layer for images (i.e. assuming input tensors with three spatial dimensions):

- Accepts a tensor of size $W_1 \times H_1 \times D_1$
- Requires 4 hyperparameters: The number of filters K , their spatial extent F , the stride with which they are applied S , and the amount of zero padding on the borders of the input, P .
- The convolutional layer produces an output volume of size $W_2 \times H_2 \times D_2$, where $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$, and $D_2 = K$.
- The number of parameters in each filter is $F * F * D_1$, for a total of $(F * F * D_1) * K$ weights and K biases. In particular, note that the spatial extent of the filters is small in space ($F \times F$), but always goes through the full depth of the input tensor (D_1)
- In the output tensor, each d -th slice of the output (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input tensor with a stride of S and then offsetting the result by d -th bias.

Biological inspiration and interpretation. The convolutional layer can be interpreted with the neuron analogy as follows. The result of a dot product with one filter at one specific location represents the (pre-activation) output of one neuron that only has connections to that specific part of the input array. Moreover, since we slide each filter over the input and use the same weights at every location we are effectively introducing a parameter sharing scheme, where neighboring neurons in one activation map all use the same weights. The result is a massive decrease in the number of parameters in each convolutional layer, which helps address overfitting. As a specific example, consider a $227 \times 227 \times 3$ input that is processed with a convolutional layer with 32 filters of size $5 \times 5 \times 3$, using padding of 2 and stride of 1. The output would in this case be $227 \times 227 \times 32$, indicating the firing of all filters at all spatial locations. This layer therefore produced $227 * 227 * 32 = 1,648,928$ outputs only using $5 * 5 * 3 * 32 + 32 = 2432$ total parameters (weights and biases). In comparison, if this was a fully connected layer with the same number of neurons in the hidden layer, we would

be using $1,648,928 * (227 * 227 * 3 + 1) = 254,904,481,664$ parameters (weights and biases) - an astronomically large number, and we'd expect to heavily overfit even if we assumed that we could compute or store the results.

Pooling layers. In addition to convolutional layers, to further control overfitting it is common to use pooling layers that decrease the size of the representation with a fixed downsampling transformation (i.e. without any parameters). In particular, the pooling layers operate on each channel (activation map) independently and downsample them spatially. A commonly used setting is to use 2×2 filters with stride of 2, where each filter computes the max operation (i.e. over 4 numbers). The result is that an input tensor is downscaled exactly by a factor of 2 in both width and height and the representation size is reduced by a factor of 4, at the cost of losing some local spatial information.

ConvNet architectures. Finally, a convolutional network is built by stacking convolutional layers and possibly introducing pooling layers to control the computational complexity of the architecture. A typical convolutional neural network architecture that processes images might take the form $[INPUT, [CONV, CONV, POOL] \times 3, FC, FC]$. Here, INPUT represents a tensor of a batch of images (e.g. $[100 \times 32 \times 32 \times 3]$ for a batch of 100 32×32 color images) CONV is a convolutional layer with 3×3 filters applied with padding of 1 and stride of 1, POOL stands for a typical 2×2 filter max pooling layer with stride of 2, and FC are fully-connected layers, where the last one computes the logits of different classes just before a softmax classifier. In this architecture the spatial size of the input is reduced by a factor of 2 in both width and height after each POOL layer, so after the third POOL layer the spatial size of the representation along width and height would be 4×4 .

2.4.3 Recurrent Neural Networks

In many practical applications the input or output spaces contain sequences. For example, sentences are often modeled as a sequence of words, where each word is encoded as a one-hot vector (i.e. a vector of all zeros except for a single 1 at the index of the word in a fixed vocabulary). A recurrent neural network (RNN) is a connectivity pattern that processes a sequence of vectors $\{x_1, \dots, x_T\}$ using a recurrence formula of the form $h_t = f_\theta(h_{t-1}, x_t)$, where f is a function that we describe in more detail below and the same parameters θ are used at every time step, allowing us to process sequences with an arbitrary number of vectors. The hidden vector h_t can be interpreted as a running *summary* of all vectors x until that time step and the recurrence formula updates the summary based on the next vector. It is common to either use $h_0 = \vec{0}$, or to treat h_0 as parameters and learn the starting hidden state. The precise mathematical form of the recurrence $(h_{t-1}, x_t) \rightarrow h_t$ varies from model to model and we describe these details next.

Vanilla Recurrent Neural Network (RNN) uses a recurrence of the form

$$h_t = \tanh \left(W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \right).$$

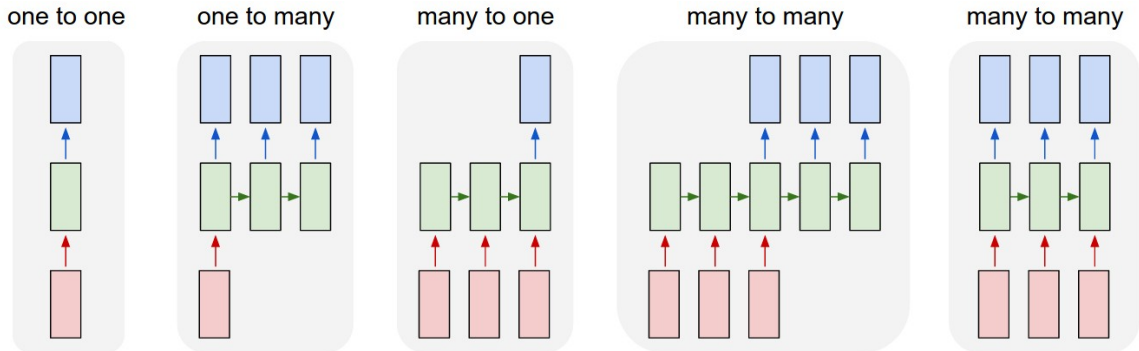


Figure 2.5: An ordinary neural network (left) might take an input vector (red), transform it through some hidden layer (green), and produce an output vector (blue). In these diagrams boxes indicate vectors and arrows indicate functional dependencies. Recurrent neural networks allow us to process sequences of vectors, for example: 1) at the output, 2) at the input, or 3) both either serially or in parallel. This is facilitated by a recurrent hidden layer (green) that manipulates a set of internal variables h_t based on previous hidden state h_{t-1} and the current input using a fixed recurrence formula $h_t = f_\theta(h_{t-1}, x_t)$, where θ are parameters we can learn.

That is, the previous hidden vector and the current input are concatenated and transformed linearly by the parameters W . Note that this is equivalent to instead writing $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1})$, where the two matrices W_{xh}, W_{hh} concatenated horizontally are equivalent to the matrix W above. These equations omit the additional bias vector for brevity. The \tanh nonlinearity can also be replaced with ReLU. If the input vectors x_t have dimension D and the hidden states dimension H then W is a matrix of size $[H \times (D + H)]$. Interpreting the equation, the new hidden states at each time step are a linear function of elements of x_t, h_{t-1} and squashed by non-linearity. The vanilla RNN has a simple form, but unfortunately, the additive interactions are a weak form of coupling [92, 101] between the inputs and the hidden states and the functional form of vanilla RNN leads to undesirable dynamics during backpropagation [6] (In particular, the gradients tend to either vanish or explode over long time periods). The exploding gradient concern can be alleviated with a heuristic of clipping the gradients at some maximum value [75], but the RNNs still suffer from the vanishing gradient problem.

Long Short-Term Memory. The LSTM [38] recurrence is designed to address the limitations of the vanilla RNN. Its recurrence formula has a form that allows the inputs x_t and h_{t-1} interact in a more computationally complex manner that includes multiplicative interactions, and the LSTM recurrence uses additive interactions over time steps that more effectively propagate gradients backwards in time [38]. In addition to a hidden state vector h_t , LSTMs also maintain a memory vector c_t . At each time step the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms. The precise form of the update is as follows:

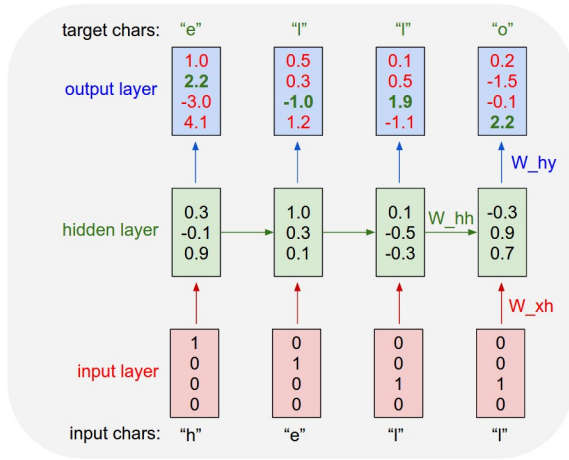


Figure 2.6: An example of using a RNN as a character-level language model. The training sequence is “hello” and the vocabulary has 4 characters: h,e,l,o. The inputs are 1-hot encodings of the characters “h,e,l,l”, and we want the RNN to predict the next character in the sequence at each time step. The RNN has a 3-dimensional hidden state (green) and there are 4 dimensions in the output vectors (blue), interpreted as the logits (i.e. scores) for the next character. The loss function and the gradient will encourage the logits of correct characters (highlighted in green) to become higher, and the other logits (in red) to become lower.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \quad \begin{aligned} c_t &= f \odot c_{t-1} + i \odot g \\ h_t &= o \odot \tanh(c_t) \end{aligned}$$

Here, the sigmoid function sigm and tanh are applied element-wise, and if the input dimensionality is D and the hidden state has H units then the matrix W has dimensions $[4H \times (D + H)]$. The three vectors $i, f, o \in \mathbb{R}^H$ are thought of as binary gates that control whether each memory cell is updated, whether it is reset to zero, and whether its local state is revealed in the hidden vector, respectively. The activations of these gates are based on the sigmoid function and hence allowed to range smoothly between zero and one to keep the model differentiable. The vector $g \in \mathbb{R}^H$ ranges between -1 and 1 and is used to additively modify the memory contents. This additive interaction is a critical feature of the LSTM’s design, because during backpropagation a sum operation merely equally distributes gradients. This allows gradients on the memory cells c to flow backwards through time uninterrupted for long time periods, or at least until the flow is disrupted with the multiplicative interaction of an active forget gate.

Example: Character-level Language Modeling. Character-level language models are a commonly studied interpretable testbed for sequence learning. In this setting, the input to the RNN is a sequence of characters from some text (e.g. “cat sat on a ”) and the network is trained to predict the next character in the sequence (“m” in this example, the first letter of “mat”). Concretely, assuming a fixed vocabulary of K characters we encode all characters with K -dimensional one-hot vectors $\{x_t\}, t = 1, \dots, T$, and feed these to the network to obtain a sequence of H -dimensional hidden vectors $\{h_t\}$. To obtain predictions for the next character in the sequence we further project the hidden states to a sequence of vectors $\{y_t\}$, where $y_t = W_y h_t$ and W_y is a $[K \times H]$ parameter matrix. These vectors are the logits of the next character (so the probabilities are obtained by passing

these through a softmax function) and the objective is to minimize the average cross-entropy loss over all targets.

2.5 Summary

In summary, a typical workflow for applying a neural network in some context looks as follows:

Data preparation. First, obtain a dataset. For the purposes of this dissertation we have assumed that the dataset is made up of a set of pairs (x, y) where x is some input example and y is a label. We then split the dataset into three folds, commonly a training, validation and test fold (common proportions could be 80%, 10%, 10% respectively). We will use the training fold for optimizing the parameters with backpropagation, the validation fold for hyperparameter optimization, and the test fold for evaluation (discussed below in more detail).

Data preprocessing. Preprocessing the data can help improve convergence of neural networks [58]. For images, common preprocessing techniques involve standardizing the data (subtracting the mean and dividing by the standard deviation individually for every input dimension of x), or at the very least subtracting the mean. It is critical to estimate these statistics *only* on the training data, and using these fixed statistics to process the validation and test data, as this appropriately simulates the deployment of the final system into a real-world application.

Architecture design. Next we must to decide on a family of architectures that we wish to explore. In our notation above, this amounts to designing the internals of the computation graph that makes up the function f . This stage is more of an art than a science, but we discuss a few common heuristics used in practice. It is common to process pixel data with convolutions and sequence data with recurrent networks. For the scale of the architecture, a very rough rule of thumb is that the full model should have approximately similar number of parameters as there are examples in the training dataset. For example the ImageNet challenge [84] dataset has 1M examples, or a small factor more if you consider data augmentation. The ConvNets we train on ImageNet usually have on order of 10M parameters and regularization techniques (such as L2 regularization, dropout, and data augmentation) are used to further constrain the model to prevent overfitting.

Optimization. A default recommendation is to use Adam [46] for optimization, with learning rates of approximately $1e^{-3}$, the coefficient of first moment of 0.9 and second moment 0.99. It is often beneficial to anneal the learning rate during the course of training by approximately a factor of 100 by the very end of training, but it is common to decay the first time only after half of the training is finished. As a good sanity check to help debug the code, it is a good idea to try to overfit a single batch of data (reaching zero training loss) before optimizing over the full training set. During optimization it is almost always a good idea to use Polyak averaging [78], where we keep track of an averaged parameter vector $\bar{\theta}$ (e.g. $\bar{\theta} = 0.999\bar{\theta} + 0.001\theta$) after every parameter update and use $\bar{\theta}$ to compute the validation performance and when saving the model checkpoint to file.

Hyperparameter optimization. The optimization with stochastic gradient can be seen as an inner loop of the optimization, while hyperparameter optimization is the outer loop that determines good values of hyperparameters that are difficult or impossible to backpropagate into (such as the learning rate, or the number of units in the hidden layers). This process consists of sampling hyperparameters from some search range (it is best to sample at random rather than along the grid [7]), optimizing the model, and evaluating the model on the validation fold. The final best model is the one that achieves the best validation performance.

Evaluation. Once we identify the best trained model (with the lowest validation loss), we evaluate the model a single time on the test set and report the performance. Consistent improvements can always be obtained by using model ensembles, which average the results of evaluating multiple models trained from different initializations or with different hyperparameters.

The remainder of this dissertation is organized around three projects that leverage this modeling machinery for connecting images and language. We now discuss the details of each project in turn.

3

Matching Images and Sentences

In this chapter we develop approaches for matching images and sentences. Concretely, we are given a finite set of images and a set of sentences. Then, given any of the sentences we would like to rank the images based on how well they depict the sentence, and conversely, given any of the images we would like to rank the sentences based on how well they describe that image. Note that this is a limited setup for image captioning because we are retrieving compatible sentences from a finite set. In the next chapter (Chapter 4) we are going to relax this constraint and generate new sentences.

We approach this problem in a data-driven manner. In particular, we develop models that consist of a neural network that takes one image and one sentence and calculates a scalar-valued score, indicating how well the image and sentence match. We then train the model end-to-end on a dataset of images captioned with sentences written by humans on Amazon Mechanical Turk (AMT), and encourage the model to assign high scores to matching image-sentence pairs, and low score otherwise (see Figure 3.1 for overview).

3.1 Related Work

The approaches developed for matching images and sentences in this chapter were originally inspired by the pioneering work of Barnard et al. [2] and Socher et al. [89] who match image regions with nouns, and Rashtchian et al., [79] and Hodosh et al. [39], who collected the first image sentence datasets and developed the first ranking techniques and evaluation criteria that we adopt in this work. Our approach was also inspired by Frome et al. [27], who associate words and images through a multimodal embedding.

Parallel to this work, a number of related approaches for grounding natural language in images using a ranking approach have been developed [49, 61, 66, 113, 111, 47]. Our work differs from many of these approaches in that our models go beyond global representations for both images and sentences and instead work on the level of image and sentence fragments. This allows us to reach

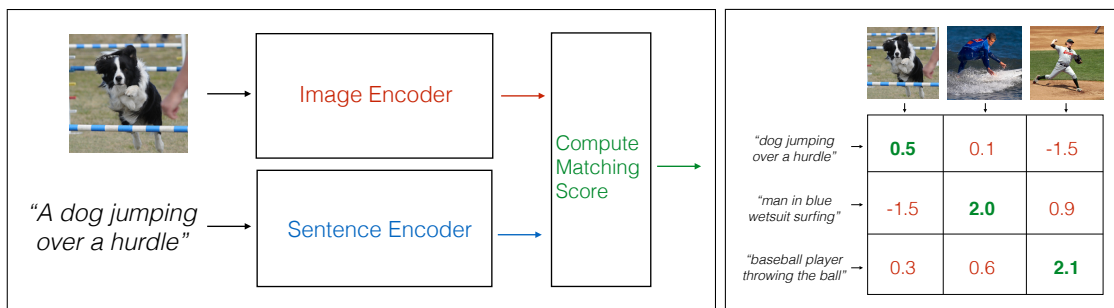


Figure 3.1: Overview of the image sentence matching model. **Left:** A single neural network that takes an image and a sentence and computes a (scalar-valued) matching score. The network consists of three components: an image encoder that process an image into a single vector v or a set of fragment vectors $\{v\}$, a sentence encoder that processes the sentence into a single vector s or a set of fragment vectors $\{s\}$, and a module that computes the score. **Right:** During training we compute pairwise scores for image-sentence pairs in a batch of training data. The loss function will encourage the true image-sentence pair scores (along the diagonal, in green) to be higher than the false image-sentence pair scores (off-diagonal, in red).

higher ranking performance and interpretable results, since we can visually inspect the sentence fragments (words) grounded in the image fragments (image regions).

Our models consist of neural networks that take images and sentences as input. Multiple approaches have been developed prior to this work for representing images and words in higher-level neural representations. On the image side, Convolutional Neural Networks (CNNs) [51, 58] have recently emerged as a powerful class of models for image classification and object detection [84]. On the sentence side, our work takes advantage of pretrained word vectors [5, 76, 69] to obtain low-dimensional representations of words. Finally, Recurrent Neural Networks have been previously used or proposed in the context of language modeling [4, 68, 92]. We build on many of these approaches and incorporate them in larger neural network architectures that model images and sentences jointly through multimodal embeddings, not in isolation.

3.2 Model

Overview. We assume that we are given a dataset of a fixed set of images, each annotated with a sentence description written by an AMT worker. We would now like to formulate the ranking problem by designing a model that takes an (image, sentence) pair and returns their compatibility score. The challenge lies in the fact that both images and sentences are complex high-dimensional objects, so relating objects between these modalities requires a model to simultaneously process and understand images, sentences, and the ability to relate objects across the two modalities.

Given an image sentence pair, models developed prior to our work [39] approached the problem in three key stages: encode the image into a vector, encode the sentence into a vector, and finally

compare the vectors across the two modalities with a loss function that encourages “true” image sentence pairs to have a high score, and “false” image sentence pairs to have a low score.

Our approach consists of first, formulating a single neural network that performs this task within the end-to-end learning paradigm and without using explicit feature representations. We show that doing so improves the performance because all components of the model share the same objective. Second, we enrich the model by reasoning about the cross-modal correspondence on the level of individual image/sentence fragments, instead of the global objects. We show that this not only leads to higher performance but also provides interpretable results.

3.2.1 Representing Images

In this section we discuss two strategies for encoding images: encoding an image I into a vector v (“global image encoding”), or into a set of vectors $\{v_r\}$ (“fragment-level image encoding”). The encoding function is designed from differentiable components to support backpropagation and contains parameters that can be learned.

Global image encoding

In encoding images into vectors we build on advances in computer vision where Convolutional Neural Networks [58] have been shown to transform raw images into powerful representations [84, 51] that support (on average) human-level performance on the ImageNet classification challenge [84]. Similar to most practical applications in computer vision, we use a ConvNet that is first pretrained on the ImageNet classification challenge. We then remove the last layer of the ConvNet which computes the 1,000 probabilities of different ImageNet classes, but keep all the other layers and parameters intact. This ConvNet can be seen as a feature extractor function $CNN_{\theta_c}(I)$, which takes image pixels I and has parameters θ_c . For example, the AlexNet [51] has approximately 60 million parameters θ_c and $CNN_{\theta_c}(I)$ is a 4096–dimensional vector. This vector is the representation after a non-linearity (e.g. ReLU in AlexNet) and immediately preceding the ImageNet classifier (which we discard).

In practice it is common to use the pretrained CNN as a fixed feature extractor and compute the features for all images in the dataset. The image encoding v then takes the form:

$$v = W[CNN_{\theta_c}(I)] + b. \quad (3.1)$$

That is, we encode the image by taking its feature vector and passing it through a linear transformation. We will train the parameters W, b and the vector v will continue to further processing in the network. It is also worth pointing out that one can easily also backpropagate through the CNN and adjust the parameters θ_c instead. This process is called *finetuning*, and while it is simple conceptually it can in many cases be much more computationally expensive (the CNN often uses most of the computation), and practically difficult compared to simply referencing a precomputed

4096-dimensional vector for any image.

Fragment-level image encoding

Following prior work [52], we observe that sentence descriptions make frequent references to objects and their attributes. Therefore, to encode images into sets of vectors we follow Girshick et al. [29] to detect objects in every image with a Region-based Convolutional Neural Network (R-CNN). The CNN is pre-trained on ImageNet [14] and finetuned on the 200 classes of the ImageNet Detection Challenge [84]. We then use the top 19 detected locations in addition to the whole image (so 20 regions in total) and compute the representations based on the pixels I_r inside each bounding box region with:

$$v_r = W[CNN_{\theta_c}(I_r)] + b, \quad (3.2)$$

where, as described in Section 3.2.1, $CNN(I_r)$ transforms the pixels inside bounding box region I_r into (e.g.) 4096-dimensional activations of the fully connected layer immediately before the classifier. The matrix W has dimensions $h \times 4096$, where h is the size of the multimodal embedding space (h takes on values of approximately 1,000 in our experiments). Every image is thus represented as a set of h -dimensional vectors $\{v_r \mid r = 1 \dots 20\}$.

Finally, notice that in this encoding we use an external, pretrained and fixed object detector to identify 19 regions that are to be encoded. This is undesirable because the resulting approach is not entirely end-to-end, but we adopt this approach to simplify the processing pipeline and keep the computational complexity low. In Chapter 5 we will instead adopt an approach where the object detector is trained jointly and fully end-to-end with the dense captioning task.

3.2.2 Representing Sentences

We saw that we can use neural networks to encode an image into a vector v or a set of fragment vectors $\{v\}$ with functions made of differentiable components and parameters. We now discuss analogous encoders for sentences. That is, we are given a sequence of words that make up a sentence and we would like to represent it as either one vector s or a set of fragment vectors $\{s\}$.

For the purposes of this section assume that each sentence is represented as a sequence of words drawn from a fixed vocabulary V of all possible words. Each word is encoded using the one-hot encoding that is all zero except for a single 1 at the index of the word in the vocabulary. That is, we will denote $\mathbb{I}_t \in R^{|V|}$ to be the one-hot encoding of the t -th word in a sentence, and a sentence is a sequence of one-hot vectors $[\mathbb{I}_1, \dots, \mathbb{I}_T]$, where T is the number of words.

As a technical side note, in practice some words might be very rare in the training data; in this case it is common to add a special UNK word (short for “unknown”) to the vocabulary and remap all rare words (e.g. occurring only 5 times in the training data) to the UNK word during preprocessing

[64].

Bag of Words

One of the simplest ways to encode a sentence is to think of each word as an individual sentence fragment, and to compute its representation by projecting the one-hot vector \mathbb{I} with a linear transformation: $s_t = W_w \mathbb{I}_t$, where W_w is a matrix of parameters to learn during backpropagation. Since \mathbb{I} is a one-hot vector, this operation effectively selects a single row of the matrix W_w . We thus call this the *word embedding matrix* (in practice common word embedding sizes are on the order of 500). If overfitting is a concern due to lack of data, the matrix W_w can also be initialized (and possibly also fixed) to word vectors obtained from other unsupervised objectives, such as word2vec [69].

The individual word representations can serve as the fragments for the sentence or we can combine them into one sentence representation through a sum (i.e. $s = \sum_t s_t$), or an average. This encoding is capable of representing what words occur somewhere in the sentence, but their spatial relationships within the sentence are lost during the encoding. For example, in encoding the sentence “a red cup on a wooden table” we would know that each word occurred, but we would lose the information about which objects the words “red” or “wooden” were describing.

n-grams. One can extend this approach by encoding not the individual words alone but bigrams (pairs), or trigrams (triples) of contiguous words. More generally we refer to these as n-gram representations. The simplest approach might be to concatenate the individual word representations into one vector $[s_t, s_{t+1}]$, or to forward this concatenated vector through one or two layers of a fully-connected neural network. This would allow the network to distinguish and represent the presence of a “red cup” instead of “red” and “cup” individually, but longer relationships within the sentence would still be lost.

Recurrent Neural Network

A recurrent neural network sentence encoder can in principle produce a representation that is a function of all words in the sentence and all of their (possibly long) relationships. A common approach is to encode every word first using a word embedding matrix and then feed each encoded word into an RNN one word at a time. Concretely, we would compute for $t = 1, \dots, T$:

$$\begin{aligned} h_0 &= \vec{0} \\ e_t &= W_w \mathbb{I}_t \\ h_t &= f(W_{hh} h_{t-1} + W_{xh} e_t + b). \end{aligned}$$

Here, the first line initializes the hidden state of the RNN at a vector of zeros, the second line

computes the word embeddings, and the last line is a vanilla RNN recurrence that strings together all word representations (in practice we could use the LSTM recurrence instead). The parameters W_w, W_{hh}, W_{xh}, b are learned with backpropagation and the non-linearity f is commonly set as tanh or ReLU. Finally, the representation s for the entire sentence could either be the last hidden state of the RNN ($s = h_T$), or the sum or average of all hidden states ($s = \sum_{t=1}^T h_t$).

It is also worth pointing out that in the above equations we’re stacking two linear transformations (W_w, W_{xh}) since $W_{xh}e_t = W_{xh}W_w\mathbb{I}_t$. This only makes sense if the word embedding size is smaller than the hidden size of the RNN H , in which case the two matrices multiplied together form a low-rank factorization of a larger $H \times |V|$ matrix (where recall that $|V|$ is the number of words in the vocabulary).

Recursive Neural Networks

Recurrent Neural Networks are a simple sentence encoder that is both rich (in that it encodes all words and their relationships) and very efficient to implement (since it scans linearly across the sentence). However, it can be argued that sentences have a compositional structure by construction and that a better suited encoder would respect this hierarchical structure during encoding. Recursive Neural Networks are a generalization of Recurrent Neural Networks in that they allow more complex encoding over arbitrary tree structure (a vanilla RNN is a special case where the tree is a chain).

Unfortunately, it is not clear how one can assign a tree for each sentence in a principled manner. Doing so with differentiable operations is nontrivial, so previous work often resorts to using off-the-shelf parsing algorithm on the sentence (e.g. either extracting a constituency parse tree or a dependency tree [13]) to determine the merging order. Another downside is that recursive networks are more difficult to parallelize because every sentence has a different merging path. However, this concern can be alleviated with careful implementation [8].

The core idea is to start with word vectors at the leafs of the tree and then recursively merge their representations all the way to the root node of the tree, which becomes the representation for the entire sentence. That is, the representation h_n for some node in the tree n is computed as $h_n = m(h_{c_n^1}, \dots, h_{c_n^q})$, where c_n^1, \dots, c_n^q are the children nodes of node n and m is a merging function. The base case are the individual words, for which the hidden representation is computed with an embedding matrix as before.

For a constituency parse tree which organizes the hierarchy of a sentence into a binary tree, the merging function in a CT-RNN [91] could take the form $h_n = f(W_l h_{c_n^1} + W_r h_{c_n^2})$, where f is a nonlinearity (e.g. tanh), W_l, W_r are parameter matrices, and $h_{c_n^1}, h_{c_n^2}$ are the left and right child representations, respectively.

A DT-RNN [90] instead merges representations over a dependency tree (we use the collapsed tree formalism of the Stanford dependency parser [13]), which organizes words in a tree structure linked by labeled edges describing the syntactic relationships. The dependency tree has the advantage that

it pushes the central content words such as the main action or verb and its subject and object to the top of the tree, giving them a larger influence on the final sentence representation. The merging formula takes a very similar form to the one provided above, except it accommodates multiple possible children per node [90]. Finally, the SDT-RNN (the first S stands for “semantic”) is an extension of the DT-RNN where every type of relationship in the dependency tree is associated with a relationship-specific weight matrix in the recursive merging function.

Bidirectional Recurrent Neural Network

Lastly, the Bidirectional Recurrent Neural Network (BRNN) [85], consists of one RNN that processes the sentence left to right and another separate RNN that processes the sentence right to left. The hidden vectors from both sides are concatenated (or averaged) to produce a representation at each word. In this way, the representation of each word is enriched by a variably-sized context around that word. The precise form of the BRNN we use looks as follows for $t = 1, \dots, T$:

$$\begin{aligned} x_t &= W_w \mathbb{I}_t \\ h_t^f &= f(W_{fx} x_t + W_f h_{t-1}^f + b_f) \\ h_t^b &= f(W_{bx} x_t + W_b h_{t+1}^b + b_b) \\ s_t &= f(W_d (h_t^f + h_t^b) + b_d). \end{aligned}$$

Note that the BRNN consists of two independent streams of processing, one moving left to right (h_t^f) and the other right to left (h_t^b). The final h -dimensional representation s_t for the t -th word is a function of both the word at that location and also its surrounding context in the sentence. A typical size of the hidden representation in our experiments ranges between 300-600 dimensions. The typical settings for the activation function f are either ReLU or tanh. The s_t function as individual fragment vectors, encoding the word \mathbb{I}_t and its context from both sides, as mediated by the two RNN streams.

3.2.3 Alignment Objective

We’ve now discussed several approaches for computing either a global representation where every image is encoded into a vector v and every sentence into a vector s , or fragment representations where every image is encoded into a set of vectors $\{v\}$ and every sentence into a set of vectors $\{s\}$. We now discuss the loss functions that can align vectors of the two modalities by thinking of these vectors as occupying a common multimodal embedding space.

Aligning Images and Sentences

Suppose that we have a training set of N images and sentences. Using the global image and sentence encoders we can compute the vectors $v_k, k = 1, \dots, N$ and $s_l, l = 1, \dots, N$, where v_k, s_l correspond if $k = l$ and do not correspond otherwise. One of the first and simplest alignment loss function one might think of uses the L2 distance loss:

$$L = \sum_{k=1}^N \|v_k - s_k\|_2^2. \quad (3.3)$$

This loss encourages the image and their corresponding sentence representations to be close. Unfortunately, this loss requires ad-hoc alternating optimization to prevent the trivial solution $v = \vec{0}, s = \vec{0}$ regardless of the input. A more natural alternative that also works significantly better in practice [90, 45] can be formulated in the max margin framework. In particular, first notice that we can use the inner product $S = v^T s$ between an image and sentence vector to be a score indicating the degree of compatibility.

Interpretation. Since $v^T s = \|v\| \|s\| \cos(\theta)$, the dot product is related to the angle between the two vectors (assuming they are both of similar lengths). Therefore, this modeling choice has the interpretation of embedding both images and sentences into a common multimodal embedding space and thinking of two objects as compatible if they point in a similar direction. An alternative concrete interpretation of the dimensions of v, s is that they can learn to *detect* related features across the two modalities. For example, the first dimension of v could be positive if there are fur-like textures in the image and the first dimension of s could be positive if “dog” or some other furry animal was mentioned somewhere in the sentence. Then if both of these dimensions are either both positive (both present) or both negative (both absent), they will positively interact in the dot product and increase the score of the match. Conversely, if fur textures are detected in the image but the sentence does not mention any fur-like animals the matching score would decrease.

Interpreting the score of the match to be $S = v^T s$, we could desire that the correct image-sentence pairs have positive scores and the incorrect ones to have negative scores. Inspired by a binary support vector machine, we could hence formulate the score as follows:

$$L(\theta) = \sum_{k=1}^N \sum_{l=1}^N \max(0, 1 - y_{kl} S_{kl}) + \lambda \|\theta\|_2^2, \quad (3.4)$$

where y_{kl} is +1 if $k = l$ and -1 otherwise. Therefore, if $k = l$ then we obtain zero loss only if the matching score is greater than 1, and if $k \neq l$ then we obtain zero loss only if the matching score is lower than -1. The regularization is added to serve the same function as it does in a binary support vector machine, where it gives rise to the max-margin property of the SVM. Note that similar to the SVM objective, it is safe to use 1 as the desired score margin instead of introducing another hyperparameter because the absolute scores can be arbitrarily inflated or shrunk by uniformly scaling

the weights by a constant and the regularization strength on the weights already encourages the weights to be low. Since the absolute scale of the weights is arbitrary it is safe to set the margin to any specific fixed constant.

The loss above is sensible but its form still imposes constraints that are stronger than what we desire. In particular, it suffices that given any image we would like the correct sentence to have a larger score than any other sentence (by a margin), and conversely, given any sentence we would like the correct image to score higher than any other image (by a margin). We can express this as:

$$L(\theta) = \sum_k \left[\underbrace{\sum_l \max(0, S_{kl} - S_{kk} + 1)}_{\text{rank images}} + \underbrace{\sum_l \max(0, S_{lk} - S_{kk} + 1)}_{\text{rank sentences}} \right] + \lambda \|\theta\|_2^2. \quad (3.5)$$

This formulation has the advantage that it precisely expresses what we desire out of the scores and nothing more. In particular, it does not impose a particular absolute scale on the scores S - they are free to vary as long as they satisfy the desired constraints and therefore as long as they correctly rank the images and the sentences.

Aligning Image Sentence Fragments

We’ve seen that if we have one image vector v and a sentence vector s then we can compute the matching score as the inner product $S = v^T s$. We now discuss the case of multiple fragment vectors $\{v\}, \{s\}$ for both images and sentences, respectively. Recall that the vectors $\{v\}$ are CNN features from important regions in the image, such as those that correspond to objects in the scene. The vectors $\{s\}$ encode the words in the sentence, enriched by their context in the sentence. Intuitively, if we have a sentence such as “dog leaps to catch a frisbee” (Figure 3.2.3), then we would like the score to be high if the entities mentioned in the sentence can be found somewhere in the image. In this example, we should be able to identify a leaping dog, the concept of catching, and a frisbee. Notice that there is an asymmetry between these two domains: anything that is mentioned in the sentence should be found somewhere in the image if the score is to be high, but the converse is not true; images may contain many other entities that might not be mentioned in the captions because they have a low saliency, but this shouldn’t prevent us from assigning a high matching score. This asymmetry motivates us to compute the matching score of the image-sentence pair as a function of the image-sentence fragment scores as follows:

$$S = \sum_t \max_i v_i^T s_t. \quad (3.6)$$

That is, for every sentence fragment s_t , $\max_i v_i^T s_t$ evaluates the best match of that fragment to any of the image fragments. Then, the \sum_t adds up the extent to which every sentence fragment

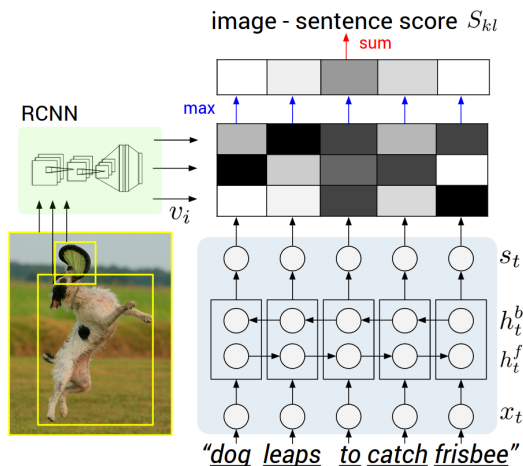


Figure 3.2: Diagram for our model. The network takes an image and a sentence and computes the image-sentence matching score S . The image is encoded using the image fragment encoding into a set of vectors $\{v\}$ - the CNN features corresponding to objects detected with an R-CNN [29] and the full image (Section 3.2.1). The sentence is encoded into fragment vectors $\{s\}$ using a bidirectional RNN (Section 3.2.2). Equation 3.6 computes the pairwise inner products between all fragments across the two modalities (the magnitudes are shown in grayscale, where white is high and black is low), and then these scores are processed with a max across columns and a sum across rows to compute the image-sentence score. The model is optimized using the loss in Equation 3.5.

aligns to something in the image to compute the final matching score.

3.3 Optimization

To optimize the model we use SGD with mini-batches of 100 image-sentence pairs and momentum of 0.9. We cross-validate the learning rate and the regularization strength. We also use dropout regularization in all layers except in the recurrent connections [108] and clip gradients elementwise at 5 (assuming that the loss/gradients are normalized across a batch but not across time). We use the AlexNet to extract features for whole images or detected objects and keep its weights fixed during optimization (i.e. we do not finetune the CNN). This is done purely as a practical simplification of the experiments.

3.4 Experiments

In the following experiments we evaluate a model that uses the image fragment encoder and the bidirectional RNN sentence fragment encoder. The image-sentence scores are computed using Equation 3.6 and the ranking loss function in Equation 3.5. We analyze this model quantitatively with respect to prior work and baselines based on some of the simpler encoders described in the previous section, and conduct qualitative study of the learned representations.

3.4.1 Data

Image-Sentence Datasets. The image-sentence datasets we use in our ranking experiments are the Flickr8K [39] and Flickr30K [106] datasets. These datasets contain 8,000 and 31,000 images respectively and each is annotated with 5 sentences by workers on Amazon Mechanical Turk (AMT).

For both datasets we use 1,000 images for validation, 1,000 for testing and the rest for training (consistent with [39, 45]).

Data Preprocessing. We convert all sentences to lowercase and discard non-alphanumeric characters. We filter words to that occur less than 5 times in the training set, which results in vocabularies with 2538 and 7414 words for Flickr8K, and Flickr30K datasets respectively.

3.4.2 Ranking Evaluation

Evaluation metric. To evaluate the model we take the test set of (withheld) images and sentences and retrieve items in one modality given a query item from the other. In particular, given an image we would like its caption to rank high in the list of all test sentences sorted by the score S (we refer to this as *Image Annotation* task), and conversely given a sentence we would like its image to rank high in the list of all test images sorted by the score S (this is an *Image Search* task). Concretely, for each item we record the rank r indicating the (integer) position of its correct associated item in the ranked list (ideally this is 1 for all images and sentences, indicating that the correct result is the top item in the list). We then report the median rank (**Med** r) across all items and Recall@K, which measures the fraction of times the correct item was found among the top K results (ideally this is 100%, and lowest possible is 0%). As a technical side note, each image has 5 sentences so in the Image Annotation task we record the best (lowest) rank among the 5 correct sentences so that the best achievable median rank remains 1. The result of these experiments can be found in Table 3.1, and example retrievals in Figure 3.3. We now highlight some of the takeaways.

Fragment representations yield higher performance. First, our fragment-based full model (“Our model: BRNN”) dramatically outperforms Socher et al. [90] (e.g. Recall@10 41.1 \rightarrow 61.4 in Image Annotation and 41.1 \rightarrow 50.5 in Image Search for Flickr30K), who trained with the same ranking loss but used a global image encoder and a global sentence encoder (a Recursive Neural Network). However, this comparison is not entirely fair because Socher et al. also use a less powerful CNN based on an autoencoder [56], while our results use the AlexNet [51] which displays superior performance on the ImageNet challenge [84].

A similar ranking loss was adopted by Kiros et al. [47] who use an LSTM sentence encoder. We list their performance with a CNN that is equivalent in power (AlexNet [51]) to the one used in this work, but similar to Vinyals et al. [98] they outperform our model with a more powerful CNN (VGGNet [88] and GoogLeNet [93] respectively). However, controlling for the strength of the CNN we again see an improvement in using fragment-level encoders and alignment scores (e.g. 50.9 \rightarrow 61.4 for Image Annotation R@10 and 46.3 \rightarrow 50.5 for Image Search R@10).

These findings are consistent with Karpathy et al. [45] (“DeFrag”), who conduct controlled experiments comparing fragment-level models to global models and find consistent improvements. Since we use different word vectors, dropout for regularization, different cross-validation ranges and larger embedding sizes, we re-implemented their loss for a fair comparison (“Our implementation of

| Model | Image Annotation | | | | Image Search | | | |
|---|------------------|-------------|-------------|------------|--------------|-------------|-------------|-------------|
| | R@1 | R@5 | R@10 | Med r | R@1 | R@5 | R@10 | Med r |
| Flickr8K | | | | | | | | |
| Kiros et al. [47] | 13.5 | 36.2 | 45.7 | 13 | 10.4 | 31.0 | 43.7 | 14 |
| Mao et al. [65] | 14.5 | 37.2 | 48.5 | 11 | 11.5 | 31.0 | 42.4 | 15 |
| Our implementation of DeFrag [45] | 13.8 | 35.8 | 48.2 | 10.4 | 9.5 | 28.2 | 40.3 | 15.6 |
| Our model: DepTree edges | 14.8 | 37.9 | 50.0 | 9.4 | 11.6 | 31.4 | 43.8 | 13.2 |
| Our model: BRNN | 16.5 | 40.6 | 54.2 | 7.6 | 11.8 | 32.1 | 44.7 | 12.4 |
| Flickr30K | | | | | | | | |
| SDT-RNN (Socher et al. [90]) | 9.6 | 29.8 | 41.1 | 16 | 8.9 | 29.8 | 41.1 | 16 |
| Kiros et al. [47] | 14.8 | 39.2 | 50.9 | 10 | 11.8 | 34.0 | 46.3 | 13 |
| Mao et al. [65] | 18.4 | 40.2 | 50.9 | 10 | 12.6 | 31.2 | 41.5 | 16 |
| Donahue et al. [18] | 17.5 | 40.3 | 50.8 | 9 | - | - | - | - |
| DeFrag (Karpathy et al. [45]) | 14.2 | 37.7 | 51.3 | 10 | 10.2 | 30.8 | 44.2 | 14 |
| Our implementation of DeFrag [45] | 19.2 | 44.5 | 58.0 | 6.0 | 12.9 | 35.4 | 47.5 | 10.8 |
| Our model: DepTree edges | 20.0 | 46.6 | 59.4 | 5.4 | 15.0 | 36.5 | 48.2 | 10.4 |
| Our model: BRNN | 22.2 | 48.2 | 61.4 | 4.8 | 15.2 | 37.7 | 50.5 | 9.2 |
| Vinyals et al. [98] (more powerful CNN) | 23 | - | 63 | 5 | 17 | - | 57 | 8 |

Table 3.1: Image-Sentence ranking experiment results. **R@K** is Recall@K (high is good). **Med r** is the median rank (low is good). In the results for our models, we take the top 5 validation set models, evaluate each independently on the test set and then report the average performance. The standard deviations on the recall values range from approximately 0.5 to 1.0.

DeFrag”). Their model uses a more complicated loss function made up of two losses (fragment loss and global loss) which we eliminate in favor of the simpler and more powerful Equation 3.6 and the ranking loss function in Equation 3.5. To examine this change in isolation we remove the BRNN and used dependency tree relations exactly as described in Karpathy et al. [45] to encode sentence fragments. The only difference between that model (“Our model: DepTree edges”) and the one of Karpathy et al. [45] is our simpler loss function and we see that the formulation developed here achieves consistent improvements (58.0 \rightarrow 59.4 and 47.5 \rightarrow 48.2 for R@10 in annotation and search respectively).

We also list the cited performance of approaches from Mao et al. [65] and Donahue et al. [18] but these models are based on an image captioning model that is trained with maximum likelihood and repurposed for ranking, while our objective directly optimizes correct ranking performance. This could partly explain our superior performance in both cases.

In summary, compared to other work that uses AlexNets, our full model shows consistent improvement that we can trace to our fragment-level treatment of images and sentences. Since fragment representations extract much more raw information from the image it is possible that the gap could be closed with much larger embedding sizes. However, this extension is non-trivial because the number of parameters in the model would also start to grow very quickly, while the fragment approach does not introduce any more parameters than a global encoding method. However, as we’ll see in the next section, the fragment-level approach does not only lead to better results but also has the benefit of providing interpretable visualizations that allow us to examine the internal computations and explicit groundings that support the model’s decisions.

BRNN outperforms dependency tree relations. Our best model uses the BRNN to encode sentence fragments. The dependency tree relations were previously shown to work better than using

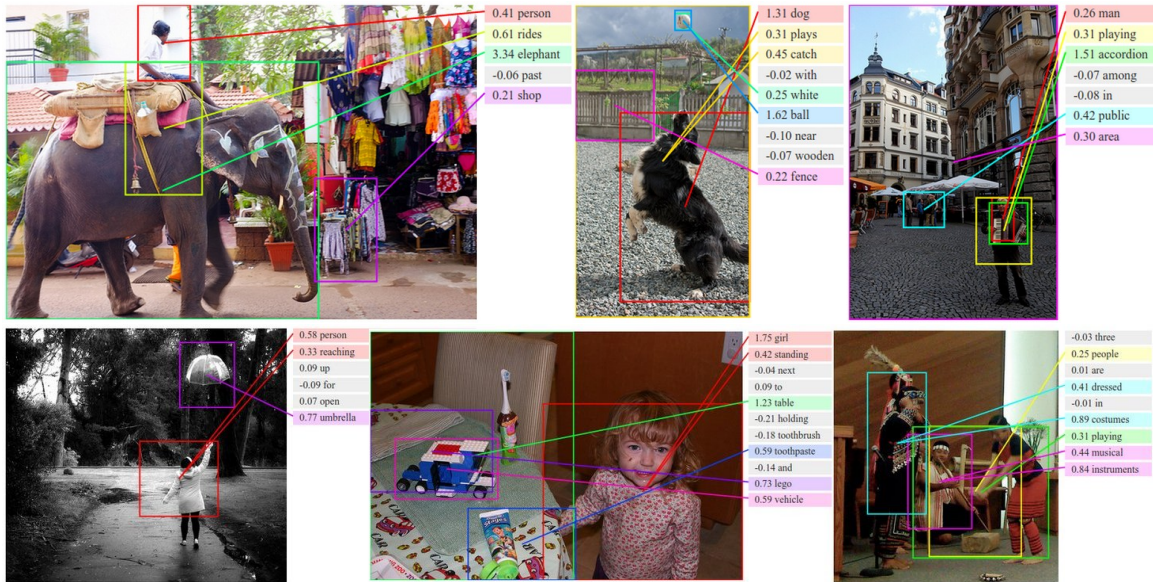



Figure 3.3: Example alignments predicted by our model. For every test image above, we retrieve the most compatible test sentence and visualize the highest-scoring region for each word and the associated scores $(v_i^T s_t)$. We hide the alignments of low-scoring words to reduce clutter. We assign each region an arbitrary color.

only individual words or bigrams [45]. In turn, we observe in our experiments that using the BRNN as an encoder further improves the performance ($59.4 \rightarrow 61.4$ and $48.2 \rightarrow 50.5$ for R@10 in annotation and search respectively). Compared to using an external dependency tree parser, the BRNN is a more end-to-end encoding approach that does not rely on computation that we cannot backpropagate through. Moreover, compared to using dependency tree edges the BRNN has the capacity to base the encoding of each fragment on more than 2 words.

3.4.3 Qualitative Evaluation

Qualitative. As can be seen from example groundings in Figure 3.3, the model discovers interpretable visual-semantic correspondences, even for small or relatively rare objects such as an “*accordion*”. These would be likely missed by models that only reason about full images. Note that one limitation of our model is that it does not explicitly handle or support counting. For instance, the last example we show contains the phrase “*three people*”. These words should align to the three people in the image, but our model puts the bounding box around two of the people. In doing so, the model may be taking advantage of the BRNN structure to modify the “*people*” vector to preferentially align to regions that contain multiple people. However, this is still unsatisfying because such spurious detections only exist as a result of an error in the RCNN inference process, which presumably failed to localize the individual people.



| Magnitude | Word | Magnitude | Word |
|-----------|-----------|-----------|--------------|
| 0.42 | now | 2.61 | kayaking |
| 0.42 | simply | 2.59 | trampoline |
| 0.43 | actually | 2.59 | pumpkins |
| 0.44 | but | 2.58 | windsurfing |
| 0.44 | neither | 2.56 | wakeboard |
| 0.45 | then | 2.54 | acrobatics |
| 0.45 | still | 2.54 | sousaphone |
| 0.46 | obviously | 2.54 | skydivers |
| 0.47 | that | 2.52 | wakeboarders |
| 0.47 | which | 2.52 | skateboard |
| 0.47 | felt | 2.51 | snowboarder |
| 0.47 | not | 2.51 | wakeboarder |
| 0.47 | might | 2.50 | skydiving |

Figure 3.4: **Left:** Flickr30K test set regions with high vector magnitude, indicating a strong influence on the image-sentence score. **Right:** This table shows the top magnitudes of vectors ($\|s_t\|$) for words in Flickr30K. Since the magnitude of individual words in our model is also a function of their surrounding context in the sentence, we report the average magnitude.

We can examine another form of a qualitative retrieval experiment in which we consider a query text snippet and then retrieve image regions across the entire test set that has the highest average score with each word in the query. We show examples of such queries in Figure 3.5. Notice that the model is sensitive to compound words and modifiers. For example, “red bus” and “yellow bus” give very different results. Additionally, it can be seen that the quality of the results deteriorates for less frequently occurring concepts, such as “straw hat”. However, we emphasize that the model learned these visual appearances of text snippets from raw data of full images and sentences, without any explicit correspondences. We have additionally published a web demo that displays our alignment results for all images in the MS COCO test set.¹

Learned region and word vector magnitudes. An appealing feature of our alignment model is that it learns to modulate the importance of words and regions by scaling the magnitude of their embedding vectors s , v . To see this, recall that we compute the image-sentence similarity between image and sentence as $S = \sum_t \max_i v_i^T s_t$.

Discriminative words. As a result of this formulation, we observe that representations of visually discriminative words such as “kayaking, pumpkins” tend to have higher magnitude in the embedding space, which translates to a higher influence on the final image-sentence scores due to the inner product. Conversely, the model learns to map stop words such as “now, simply, actually, but” near the origin, which reduces their influence. The table in Figure 3.4 shows the top few words with highest and lowest magnitudes $\|s_t\|$.

Discriminative regions. Similarly, image regions with discriminative entities are assigned vectors of higher magnitudes by our model. This can be interpreted as a measure of visual saliency, since these regions would produce large scores if their textual description was present in a corresponding sentence. We show some regions with high magnitudes in Figure 3.4 (left). Notice the common occurrence of often described and easily visually identifiable regions such as balls, bikes, helmets.

¹<http://cs.stanford.edu/people/karpathy/deepimagesent/rankingdemo/>



Figure 3.5: Examples of highest scoring regions for queried snippets of text, on 5,000 images of MS COCO test set images.

3.5 Conclusions

We have developed a model that can match images and sentences. The model consists of a single neural network that takes image-sentence pairs and is trained end-to-end with a ranking objective over image-sentence datasets. Our evaluation shows strong performance compared to previous methods and baseline approaches that use simpler (global) image and sentence encoders. Our qualitative experiments also show that the specific structure of our architecture allows us to inspect the model’s predictions and gain an understanding of its predictions.

One of the limitations of the model is that we assumed an external object detector (RCNN) to extract salient regions and their CNN features. In Chapter 5 we describe a model that includes the object detection module in the model architecture. Another limitation is that we model both the image and the sentence as a set of vectors, discarding a lot of information such as the spatial position of each bounding box in the image. Therefore, the model would not be able to distinguish between sentences such as “a dog on the left of a person” and “a dog on the right of a person”.

4

Generating Image Captions

In the previous chapter we've developed a model that can annotate images with sentences by iteratively checking a matching score with one image and many candidate sentences written by humans. This represents a step up from annotating an image with a single discrete category, but the approach suffers from multiple limitations:

1. We are restricted to only a finite collection of sentences and the algorithm is incapable of mixing and matching the individual pieces of sentences and generalizing to new concepts.
2. The approach is inefficient because in order to label an image we must iterate over many different sentences and check their score one by one.
3. The approach is unsatisfying; when humans describe images they do so efficiently and directly; it doesn't feel as though we are internally matching thousands of full-formed candidate descriptions against the image.

In this chapter we develop a model that can take an image and directly generate its sentence description. The model is based on a novel combination of a Convolutional Neural Network to process the image and a Recurrent Neural Network Language Model that is conditioned on the image information. Unlike previous pioneering work in the image captioning literature [26, 52] our model is trained end-to-end on raw data and with few explicit assumptions. The model allows us to sample likely descriptions of images in a simple, efficient feedforward process. Our experiments show that this approach allows us to outperform retrieval baselines similar to the ones in the previous chapter that are constrained to a finite collection of sentences to choose from.

4.1 Related Work

A number of approaches pose the task of generating descriptions for images as a retrieval problem, where the most compatible annotation in the training set is transferred to a test image [26, 39, 41, 73, 90], or where training annotations are broken up and stitched together [53, 55, 60]. Several approaches generate image captions based on fixed templates that are filled based on the content of the image [1, 21, 26, 35, 52, 103, 104] or generative grammars [105, 70], but this approach limits the variety of possible outputs.

More closely related to us is the approach of Kiros et al. [47, 48] who developed a log-bilinear neural network model that generates captions based on a finite length context window. Several approaches that alleviate the finite context constraint have been developed in parallel with this work based on a Recurrent Neural Network language model [10, 18, 25, 65, 98]. Our RNN is simpler than most of these approaches but also suffers slightly in performance. We quantify this comparison in our experiments. Finally, Recurrent Neural Networks have been previously used or proposed in the context of language modeling [4, 68, 92], but we additionally condition these models on images.

4.2 Model

In this section we assume an input set of images, each annotated with a sentence description. The key challenge is in the design of a model that can predict a variable-sized sequence of words for each image.

RNN language models. We can look to Natural Language Processing (NLP) literature for inspiration on how to tackle this problem. In particular, language modeling is a common task in NLP where one wants to represent probability distributions over sentences (treated as sequences of words). In particular, we would like to model the joint distribution $p(x_1, \dots, x_T)$ where x_1, \dots, x_T are words. Notice that we can factor this joint distribution into a product of marginals: $p(x_1, \dots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_T|x_1, \dots, x_{T-1}) = \prod_{t=1}^T p(x_t|x_1, \dots, x_{t-1})$. Therefore, we can represent distributions over sentences by modeling the conditional probability of a word given all the previous words in the sequence. Previous work has relied on Recurrent Neural Networks (RNNs) [22, 68, 92], to model this conditional probability since it can (in principle) capture dependencies of arbitrary lengths between the words, unlike the more traditional n -gram models that only condition a word based on the last n words before it, usually with a lookup table based on counts and smoothing techniques for handling outliers.

Multimodal RNN language model. We explore a simple but effective extension that additionally conditions the language model’s generative process on the content of an input image. We call this model a Multimodal RNN. During training our Multimodal RNN takes the raw image pixels I and a sequence of input vectors (x_1, \dots, x_T) . The vectors x_t could be linear projections of one-hot input vectors using a word embedding matrix that can be learned with backpropagation, or if the

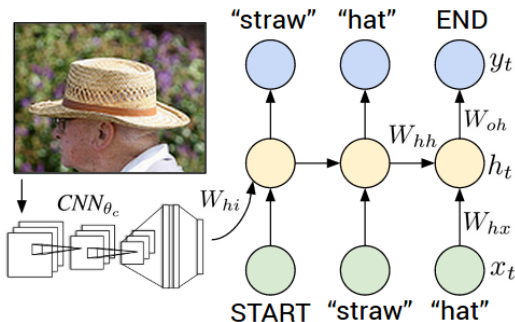


Figure 4.1: Diagram of the Multimodal Recurrent Neural Network model. Each arrow indicates a functional dependence. The RNN takes a word vector (green) and the context vector from previous time steps (yellow) and defines a distribution over the next word in the sentence (blue vectors). The RNN is conditioned on the image information at the first time step (the arrow annotated with the weight matrix W_{hi}). START and END are special tokens denoting the beginning and the end of the sentence.

data size is a concern they can be set according to a separate criterion (e.g. word2vec [69]). The Multimodal RNN then computes a sequence of hidden states (h_1, \dots, h_T) and a sequence of output vectors (y_1, \dots, y_T) by iterating the following recurrence relation for $t = 1$ to T :

$$b_v = W_{hi}[CNN_{\theta_c}(I)] \quad (4.1)$$

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h + \mathbb{1}\{t = 1\} \odot b_v) \quad (4.2)$$

$$y_t = W_{oh}h_t + b_o. \quad (4.3)$$

In the equations above, $W_{hi}, W_{hx}, W_{hh}, W_{oh}$ and b_h, b_o are learnable parameters, $CNN_{\theta_c}(I)$ is the last feature layer of a CNN (after non-linearity), f is the activation function (we use ReLU), and the hidden state h_0 we initialize to a zero vector. The output vectors y_t hold the logits of all words in the dictionary. The vocabulary also contains an additional special END token that corresponds to the period at the end of the sentence, and which we expect the RNN to always predict as the last word in any sentence. Note that we provide the image context vector b_v to the RNN only at the first iteration (modulated by a multiplicative interaction with the delta function $\mathbb{1}\{t = 1\}$), which we found to work better than providing this information at each time step (possibly due to easier overfitting). Note that the RNN has to juggle two tasks simultaneously: 1) it has to remember the image information in its hidden state and 2) it has to use its hidden state to maintain internal variables that allow it to track context and predict the desired sequences. For the purposes of achieving 1), note that the RNN has the capacity to use some dimensions of h to store the image information by learning identity recurrent connections for these units, which would exactly preserve the image information. E.g. to exactly store and maintain the first hidden state the RNN would learn the first row of W_{hh} to be $[1, 0, 0, \dots, 0]$, and the first row of W_{hx} to be all zero. A typical size of the hidden layer of the RNN in our experiments is 512 neurons.

RNN training. The training proceeds as follows (refer to Figure 4.1): We set $h_0 = \vec{0}$, x_1 to a special START vector, and the desired label for y_1 to be the first word in the ground truth sequence.

Analogously, at the second time step $t = 2$, we set x_2 to the word vector of the first word and expect the network to predict the second word, etc. Finally, on the last step when x_T represents the last word, the target label is set to a special END token. The cost function is to maximize the log probability assigned to the target labels (i.e. the Softmax classifier, or the *cross-entropy* loss).

Finally, note that the number of time steps T is one greater than the number of words in the ground truth caption for the image due to the offset introduced by the START and END tokens. For example, a caption with 7 words would require $T = 8$ applications of the recurrence. In practice we upper bound T to 16. When constructing mini-batches of data during optimization we keep track of the length of each individual caption in the minibatch. We then forward the RNN for 16 timesteps on the entire minibatch regardless of the lengths of the individual sentences (this is done to preserve the efficiency of parallel computation). However, during backpropagation we keep track of the index of last prediction for each example in the minibatch and backpropagate accordingly. An alternative and possibly faster solution is to pad sequences with END tokens and expect the RNN to always predict END token if the END token is fed in as the input. This way there is no need to track the indices of the last word for each example in a minibatch.

RNN at test time. To predict a caption for a test image we compute the image representation b_v , set $h_0 = \vec{0}$, x_1 to the START vector, compute the logits for the first word in the caption, y_1 , and pass this through the softmax function to obtain a probability distribution for the first word in the sequence. We normalize the distribution and sample a word (or pick the argmax), set x_2 to be its embedding vector, and repeat this process until the END token is generated. In practice we found that beam search decoding [110] (e.g. with beam size 3) can improve results since it normally produces a more globally likely caption for an input image, which might otherwise not be produced in a greedy manner one word at a time. Beam search is implemented by keeping track of some fixed number of the top most likely sequences so far at each time step. Therefore, if the beam size is 1 then this corresponds to greedily appending the single most likely word at that time step to the prediction and proceeding forward.

4.3 Optimization

The Multimodal RNN is difficult to optimize partly due to the word frequency disparity between rare words and common words (e.g. the END token occurs in every sentence and the word “a” in nearly every sentence, while the word “toothbrush” but might only occur 5 times in total). We achieved good results with **RMSprop** [96], which is an adaptive step size method that scales the update of each weight by a running average of its gradient norm. We also experimented with SGD, SGD+Momentum, Adadelta and Adagrad, but found these to work worse. In later experiments we also used Adam, which slightly outperformed RMSProp. We found that **clipping the gradients** [75] (we only experimented with simple per-element clipping) at an appropriate value provided consistent

but small improvements. Since the distribution of the words in English is highly non-uniform, the model spends the first few iterations mostly learning the biases for the Softmax classifier such that it is predicting every word at random with the appropriate dataset frequency. We found that we could obtain faster convergence early in the training (and nicer-looking loss curves) by explicitly **initializing the biases** of all words in the dictionary (in the Softmax classifier) to log probability of their occurrence in the training data. Therefore, with small weights and biases set appropriately the model right away predicts word at random according to their chance distribution. Lastly, we experimented with initializing word representations x_i with word2vec vectors [69], but found that it was sufficient to train these vectors from random initialization without changes in the final performance. Moreover, we found that the word2vec vectors have some unappealing properties when used in multimodal language-visual tasks. For instance, all colors (e.g. red, blue, green) are clustered nearby in the word2vec representation because they are relatively interchangeable in most language contexts. However, their visual instantiations are very different.

4.4 Experiments

Datasets. The image captioning datasets we use in our experiments are the Flickr8K [39], Flickr30K [106] and MS COCO [62]. These datasets contain 8,000, 31,000 and 123,000 images respectively and each is annotated with 5 sentences using Amazon Mechanical Turk (AMT). For Flickr8K and Flickr30K, we use 1,000 images for validation, 1,000 for testing and the rest for training (consistent with [39, 45]). For MS COCO we use 5,000 images for both validation and testing.

Data Preprocessing. We convert all sentences to lowercase and discard non-alphanumeric characters. We filter words to those that occur at least 5 times in the training set, which results in 2538, 7414, and 8791 words for Flickr8k, Flickr30K, and MS COCO datasets respectively.

We now evaluate the ability of our RNN model to describe images. We use the VGGNet [88] as the CNN. The word vector embedding size is 512 and the size of the hidden layer in the RNN is 512. As evaluation metrics we use the BLEU [74], METEOR [15] and CIDEr [97] scores computed with the `coco-caption` code [9].¹ Each one of these methods evaluates a *candidate* sentence by measuring how well it matches a set of five *reference* sentences written by humans.

Hyperparameters. We use an RNN with 512 units in the hidden layer. Even with this relatively low amount of units we find that the model is prone to overfitting, and we experimented with introducing a dropout layer between the hidden state and the next step predictions with a probability of dropping units from 0 to as high as 0.9, without significant performance deterioration (dropout of about 0.5 worked well). The initial learning rate for Adam was $4e^{-4}$, but we found that the performance was much less sensitive to the learning rate with Adam than with simple SGD. Adam’s α was set to 0.8 and β to 0.999, but these settings were not particularly sensitive. We also

¹<https://github.com/tylin/coco-caption>

| Model | Flickr8K | | | | Flickr30K | | | | MS COCO 2014 | | | |
|-----------------------|----------|------|------|------|-----------|------|------|------|--------------|------|---------|-------|
| | B-1 | B-2 | B-3 | B-4 | B-1 | B-2 | B-3 | B-4 | B-1 | B-4 | ME-TEOR | CIDEr |
| Nearest Neighbor | — | — | — | — | — | — | — | — | 48.0 | 10.0 | 0.157 | 0.38 |
| Mao et al. [65] | 58 | 28 | 23 | — | 55 | 24 | 20 | — | — | — | — | — |
| Google NIC [98] | 63 | 41 | 27 | — | 66.3 | 42.3 | 27.7 | 18.3 | 66.6 | 24.6 | — | — |
| LRCN [18] | — | — | — | — | 58.8 | 39.1 | 25.1 | 16.5 | 62.8 | — | — | — |
| MS Research [25] | — | — | — | — | — | — | — | — | — | 21.1 | 0.207 | — |
| Chen and Zitnick [10] | — | — | — | 14.1 | — | — | — | 12.6 | — | 19.0 | 0.204 | — |
| Our model | 57.9 | 38.3 | 24.5 | 16.0 | 57.3 | 36.9 | 24.0 | 15.7 | 62.5 | 23.0 | 0.195 | 0.66 |

Table 4.1: Evaluation of full image predictions on 1,000 test images. **B-n** is BLEU score that uses up to n-grams. High is good in all columns. For future comparisons, our METEOR/CIDEr Flickr8K scores are 16.7/31.8 and the Flickr30K scores are 15.3/24.7.

| Word query | Nearest words | Word query | Nearest words |
|---------------|--|----------------|--|
| car | van, atv, snowmobile, minivan, figs, motorcyclist | several | multiple, couple, many, numerous |
| street | highway, alleyway, sidewalk, roadway, freeway | horse | pony, ox, dirtbike, horseback, giraffee |
| bike | bicycle, motorbike, moped, motorcycle | two | 2, five, four, three, multiple, 3 |
| looks | stares, look, staring, starring, gazing, looked | jumps | leaping, leaps, jumping, analog |
| red | purple, maroon, burgundy, beige, jalapenos | truck | firetruck, van, scooter, panels, minivan |
| ocean | sea, pond, waters, crashing, building, parasailers | below | underneath, beneath, inspects |

Table 4.2: Learned word representations. Our model automatically arranges all words in an embedding, where nearby words have similar affects on the model. This table shows some example word queries (in bold) and to the right of them their closest words in the learned embedding in order.

experimented with a model that received the image information at every time step instead of just the first time step and observed lower BLEU-4 scores by about 2 points.

Multimodal RNN outperforms retrieval baseline. Our first comparison is to a nearest neighbor retrieval baseline. Here, we annotate each test image with a sentence of the most similar image from the training set, as determined by L2 distance based on the VGGNet [88] features. Table 4.1 shows that the Multimodal RNN confidently outperforms this retrieval method. It is worth noting that a more sophisticated nearest neighbor approach based on multiple neighbors and a consensus voting strategy can be very competitive with state of the art CNN-RNN approaches similar to the one explored in this work [16]. However, unlike retrieval approaches based on nearest neighbor techniques or ranking methods as those explored in the previous chapter, the parametric approach here takes only a fraction of a second to evaluate per image.

Comparison to other work. Several related models have been developed in parallel to this work, which we include in Table 4.1 for comparison. Most similar to our model is the model of Vinyals et al. [98]. Unlike this work where the image information is communicated through a bias term on the first time step of the RNN, they incorporate the image information as a first word. They also use a more powerful but more complex sequence learner (LSTM [38]), a different CNN (GoogLeNet [93]), and report they results of a model ensemble while we report the results of a single model. Donahue et al. [18] use a 2-layer factored LSTM (similar in structure to the RNN in Mao et al. [65]). Both models appear to work worse than ours, but this is likely in large part due to their use of the less powerful AlexNet [51] as the convolutional network. Compared to these approaches,

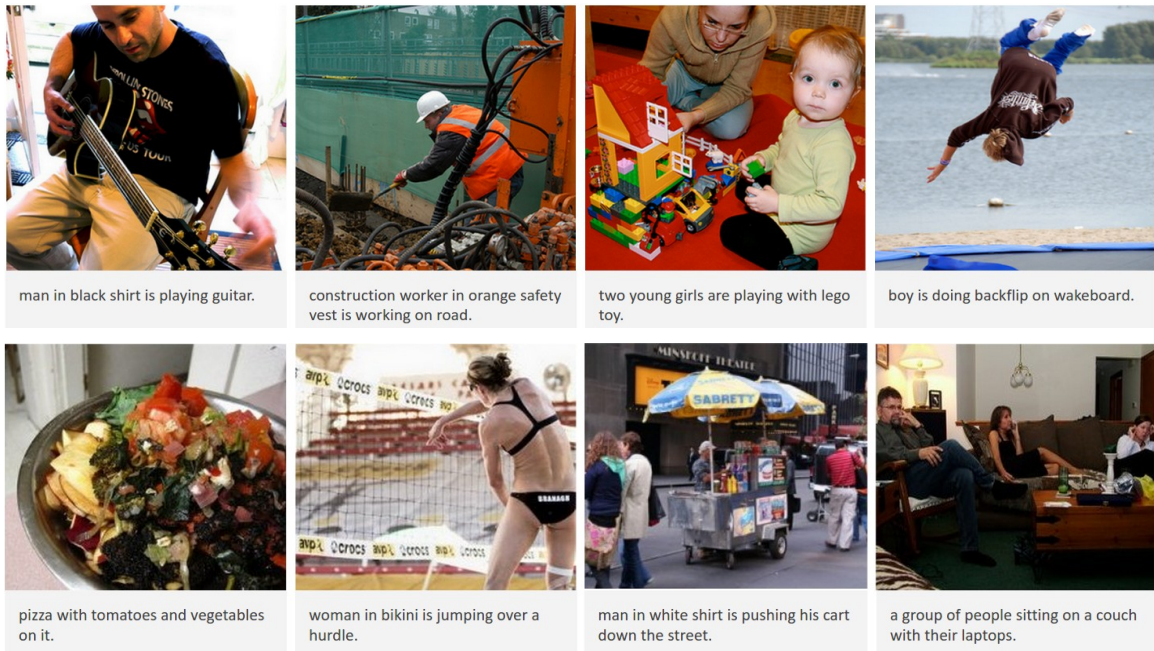


Figure 4.2: Example sentences generated by the Multimodal RNN for test images.

our model prioritizes simplicity and speed at a slight cost in performance.

Learned word representations. The model automatically arranges the words in an embedding space using a word embedding matrix that is trained with backpropagation. We can inspect the learned representations to gain an understanding of its layout. Table 4.2 shows some example words and their nearest neighbors in the learned embedding representation, as determined by taking the inner product of their word embedding vectors. Words that are close in this table have a similar effect on the model’s further predictions. For instance, we see that the model has learned that the words *cat*, *van*, *atv*, *snowmobile*, *minivan* all have similar meanings and effects. The model also learned synonyms of *several*, *jumps*, *below*, *red*, etc. Notice that these embeddings are learned from scratch on raw data, so the model must also figure out that both *staring* and its incorrectly spelled form *starring* are both similar to the word *looks*.

Qualitative. The model generates sensible descriptions of images (see Figure 4.2) but commonly makes subtle mistakes. For example, the man’s shirt on the first image is very dark blue, the construction worker does not appear to be working on a road (but it is difficult to tell), etc. We were curious to quantify the extent to which the model creates novel descriptions. For example, the first prediction “*man in black shirt is playing a guitar*” does not appear in the training set. However, there are 20 occurrences of “*man in black shirt*” and 60 occurrences of “*is paying guitar*”, which the model may have composed to describe the first image. The model often gets the right gist of the scene, but sometimes guesses specific fine-grained words incorrectly.

We find one example result (“*woman in bikini is jumping over a hurdle*”) to be especially illuminating. This sentence does not occur in the training data. Our general qualitative impression of the model is that it learns certain templates, e.g. [`<noun>in <noun>is <verb>in <noun>`], and then fills these in based on textures in the image. In this particular case, the model has first correctly identified a woman in bikini jumping. Then, the volleyball net has the visual appearance of a hurdle and the phrase “jumping over a hurdle” occurs often in the training data (usually featuring dogs), which together may have caused the model to insert this substring into one of its learned sentence templates.

Caption statistics comparisons. We can get a deeper understanding of the generated captions by considering the statistics of generated sentences over the entire test set and comparing them to the statistics of human-written captions.

First, in general, we find that a relatively large portion of generated sentences (60% with beam size 7) can be found in the training data. This fraction decreases with lower beam size; For instance, with beam size 1 this falls to 25%, but the performance also deteriorates (e.g. from 0.66 to 0.61 CIDEr). In our experiments we found that beam size of 2 worked best on the validation data (followed closely by beam size 3), where 46% of the generated sentences can be found in the training data. Note that the training dataset contains a large amount of sentences (a bit more than half a million). We find that the generated captions have a much lower variety than human captions. In the entire dataset of training sentences we find that the ratio of unique sentences to all sentences is 95.7% for human captions (i.e. barely any captions are repeats), while this ratio is 30% for the generated captions. That is, of all predictions on the test set of 40,000 test images we find only 12,000 unique captions.

To further quantify the discrepancy in the variety, we parsed all generated and human-written sentences with part of speech tags. For example, the sentence “a man sitting on a couch with a dog” has the part of speech tags `[DT, NN, VBG, IN, DT, NN, IN, DT, NN]`, where for example `DT` are determiners (“a”, “the”, etc.), `NN` are nouns, `VB*` are verbs, etc. We can then analyze the variety of part of speech tag forms. In particular, we find that the most common form of the generated captions is `[DT NN IN DT NN CC DT NN]`, which accounts for 4.5% of all generated sentences (an example is “a plate with a sandwich and a salad”). This is followed by `[DT NN VBZ VBG IN DT NN IN DT NN]` that accounts for 3.2% of generated captions (an example is “a boat is sailing in the water near a city”). Overall, we find that the uniqueness ratio is very low for the generated sentences (8%) while it is much higher for human-written sentences (73%). Moreover, the top 10 of tag sequences account for 20% of the generated captions and the top 100 account for 53% of the generated captions. For human-written captions the top 10 tag sequences only account for 2% of the captions, and the top 100 for 8% of the captions. This again shows that humans generate significantly more varied sentence structures.

Lastly, we can study this discrepancy on the level of words. As mentioned, the vocabulary used

during training is almost 10,000 words. However, we find that the network’s predictions across the entire test set only contain 1,079 unique words, indicating that 9,000 words are not learned, likely because they are very rare in the training data. Moreover, 82% of the generated words come from a collection of only 100 words, while the top 100 words used by humans only form 64% of all words used in human captions. This indicates that the vocabulary used by humans is much richer and varied, while the network uses a vocabulary of only 100 words 82% of the time, and commands up to about 1,000 words in total.

We would like to emphasize that the networks only train on about 100,000 images, which is a very small collection compared to ImageNet and its 1,200,000 images. In addition, the MS COCO dataset does not have a large variety of images and qualitative assessment suggests that very large portions of images contain giraffes, elephants, zebras and toilets. Therefore, the networks are not faced with a large quantity or variety during training and we expect that further improvements can be obtained by scaling our image-sentence datasets along the size and variety dimensions.

4.5 Conclusions

We developed a model that can generate image captions. The model is a Recurrent Neural Network Language Model conditioned on image information through an added interaction with the representation produced by a Convolutional Neural Network. The entire model can be optimized end-to-end on raw image-sentence datasets. Quantitative experiments demonstrate that the model can caption images better than an approach based on a retrieval method that is only constrained to a finite collection of sentences. Qualitative experiments display sensible properties of the learned word representations and overall appealing and sometimes amusing captions. Statistical analysis of the captions suggests that the results lack in variety compared to the variety of captions that humans produce and we expect that further advances can be made both by more sophisticated models and by scaling up the size and variety of the image-sentence datasets.

Although our results are encouraging, the model is subject to multiple limitations. First, the model can only generate a description of one input array of pixels at a fixed resolution. A more sensible approach might be to use multiple saccades around the image to identify all entities, their mutual interactions and wider context before generating a description. Additionally, the RNN receives the image information only through additive bias interactions, which are known to be less expressive than more complicated multiplicative interactions [92, 38]. Finally, the model does not provide an easy way to do image retrieval based on captions, as seen in the ranking task of Chapter 3. It is possible to search for images that would have been likely to generate the given caption, but the model’s objective was not explicitly designed or trained for this testing protocol.

5

Dense Image Captioning

In Chapter 3 we saw that we can label images with sentences chosen from a finite set of human-written captions. In Chapter 4 we then introduced a model that generate new sentences without being constrained to merely choosing from a given collection of sentences. This approach is much more computationally efficient and satisfying philosophically because it can learn to compose snippets of captions it has seen in the past to generate novel sentences, but it is still subject to several limitations. In particular, if we feed a complex image to an image captioning model the network may have to fall back on relatively vague and generic descriptions in an effort to describe the full image (e.g. “a group of people sitting in an office”; see Figure 5.1, left). In this chapter we develop DenseCap, a model that relaxes this constraint and can simultaneously both detect and describe everything in a visual scene (Figure 5.1, right), including the full image as merely a special case.



Figure 5.1: **Left:** An image captioning model is forced to fall back on a generic caption to describe the image in one sentence. **Right:** The proposed DenseCap model can both localize and describe all elements of the visual scene, including the full image as a special case.

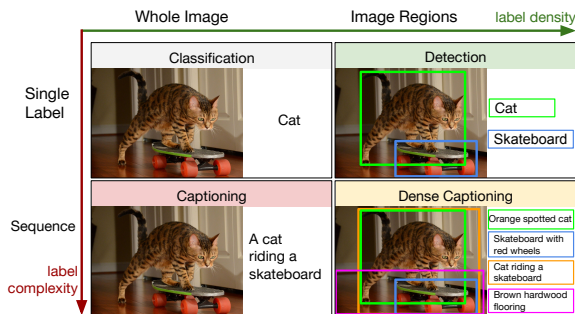


Figure 5.2: The Dense Captioning task (bottom right) simultaneously combines the challenges of object detection (high label density, x-axis) and image captioning (high label complexity, y-axis). That is, the model must both detect and describe all salient aspects of the visual scene. Importantly, a dense captioning model can also allow the converse task of using a query description to search an image collection and identify matching regions.

Our ability to effortlessly point out and describe all aspects of an image relies on a strong semantic understanding of a visual scene and all of its elements. However, despite numerous potential applications, this ability remains a challenge for our state of the art visual recognition systems. In the last few years there has been significant progress in image classification [84, 51, 109, 93], where the task is to assign one label to an image. Further work has pushed these advances along two orthogonal directions: First, rapid progress in object detection [86, 29, 94] has identified models that efficiently identify and label multiple salient regions of an image. Second, recent advances in image captioning [9, 65, 43, 98, 102, 18, 10] have expanded the complexity of the label space from a fixed set of categories to sequence of words able to express significantly richer concepts.

However, despite encouraging progress along the label density and label complexity axes, these two directions have remained separate. In this work we take a step towards unifying these two inter-connected tasks into one joint framework. First, we introduce the dense captioning task (see Figure 5), which requires a model to predict a set of descriptions across regions of an image. Object detection is hence recovered as a special case when the target labels consist of one word, and image captioning is recovered when all images consist of one region that spans the full image.

Additionally, we develop a Fully Convolutional Localization Network architecture (FCLN) to address the dense captioning task. Our model is inspired by recent work in image captioning [98, 43, 65, 18, 10] in that it is composed of a Convolutional Neural Network followed by a Recurrent Neural Network language model. However, drawing on work in object detection [81], our second core contribution is to introduce a new dense localization layer. This layer is fully differentiable and can be inserted into any neural network that processes images to enable region-level training and predictions. Internally, the localization layer predicts a set of regions of interest in the image and then uses bilinear interpolation [40, 32] to smoothly extract the activations inside each region.

We evaluate the model on the large-scale Visual Genome dataset, which contains 94,000 images and 4,100,000 region captions. Our results show both performance and speed improvements over approaches based on previous state of the art. We make our code and data publicly available to support further progress on the dense captioning task.

5.1 Related Work

Our work builds on recent work in object detection, image captioning, and soft spatial attention that allows downstream processing of particular regions in the image.

Object Detection. Our core visual processing module is a Convolutional Neural Network (CNN) [58, 51], which has emerged as a powerful model for visual recognition tasks [84]. The first application of these models to dense prediction tasks was introduced in R-CNN [29], where each region of interest was processed independently. Further work has focused on processing all regions with only single forward pass of the CNN [36, 30], and on eliminating explicit region proposal methods by directly predicting the bounding boxes either in the image coordinate system [94, 23], or in a fully convolutional [63] and hence position-invariant settings [86, 81, 80]. Most related to our approach is the work of Ren et al. [81] who develop a region proposal network (RPN) that regresses from anchors to regions of interest. However, they adopt a 4-step optimization process, while our approach does not require training pipelines. Additionally, we replace their RoI pooling mechanism with a differentiable, spatial soft attention mechanism [40, 32]. In particular, this change allows us to backpropagate through the region proposal network and train the whole model jointly.

Image Captioning. Several pioneering approaches have explored the task of describing images with natural language [2, 52, 26, 72, 89, 90, 54, 41]. More recent approaches based on neural networks have adopted Recurrent Neural Networks (RNNs) [99, 38] as the core architectural element for generating captions. These models have previously been used in language modeling [4, 31, 68, 92], where they are known to learn powerful long-term interactions [44]. Several recent approaches to Image Captioning [65, 43, 98, 18, 10, 47, 25] rely on a combination of RNN language model conditioned on image information. A recent related approach is the work of Xu et al. [102] who use a soft attention mechanism [11] over regions of the input image with every generated word. Our approach to spatial attention is more general in that the network can process arbitrary affine regions in the image instead of only discrete grid positions in an intermediate conv volume. However, for simplicity, during generation we follow Vinyals et al. [98], where the visual information is only passed to the language model once on the first time step.

Finally, the metrics we develop for the dense captioning task are inspired by metrics developed for image captioning [97, 15, 9].

5.2 Model

Our goal is to design an architecture that jointly localizes regions of interest and then describes each with natural language. The primary challenge is to develop a model that supports end-to-end training with a single step of optimization, and both efficient and effective inference. Our proposed architecture (see Figure 5.3) draws on architectural elements present in recent work on object detection, image captioning and soft spatial attention to simultaneously address these design

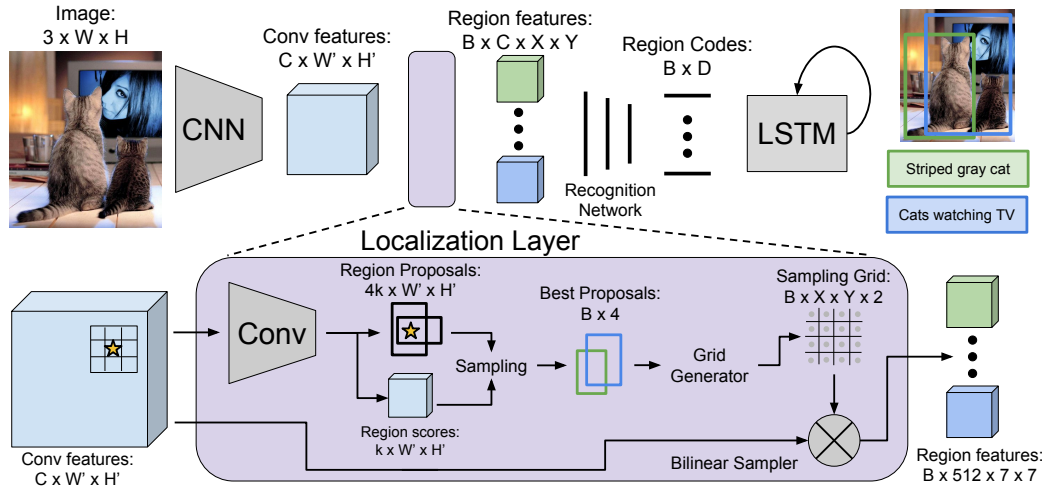


Figure 5.3: Model overview. An input image is first processed a CNN. The Localization Layer proposes regions and smoothly extracts a batch of corresponding activations using bilinear interpolation. These regions are processed with a fully-connected recognition network and described with an RNN language model. The model is trained end-to-end with gradient descent.

constraints.

In Section 5.2.1 we first describe the components of our model. Then in Sections 5.2.2 and 5.2.3 we address the loss function and the details of training and inference.

5.2.1 Model Architecture

Convolutional Network

As the core visual processing module we use the VGG-16 architecture [88] due to its state-of-the-art performance on the ImageNet challenge [84]. The network consists of 13 layers of 3×3 convolutions interspersed with 5 layers of 2×2 max pooling. We remove the final (fifth) pooling layer, so an input image of shape $3 \times W \times H$ gives rise to a tensor of features of shape $C \times W' \times H'$ where $C = 512$, $W' = \lfloor \frac{W}{16} \rfloor$, and $H' = \lfloor \frac{H}{16} \rfloor$ (16 because we only have 4 max-pooling layers remaining, each halving the spatial resolution). This tensor (in light blue in Figure 5.3) then feeds into the localization layer (purple), discussed next.

Fully Convolutional Localization Layer

The localization layer receives an input tensor of activations, identifies spatial regions of interest and smoothly extracts a fixed-sized representation from each region. Our approach is based on that of Faster R-CNN [81], but we replace their RoI pooling mechanism [30] with bilinear interpolation [40], allowing our model to propagate gradients backward through the coordinates of predicted regions.

This modification also opens up the possibility of predicting affine or morphed region proposals instead of rectangular bounding boxes [40], but we leave these extensions to future work.

Inputs/outputs. The localization layer accepts a tensor of activations of size $C \times W' \times H'$. It then internally selects B regions of interest and returns three output tensors giving information about these regions:

1. **Region Coordinates:** A matrix of shape $B \times 4$ giving bounding box coordinates for each output region.
2. **Region Scores:** A vector of length B giving a confidence score for each output region. Regions with high confidence scores are more likely to correspond to ground-truth regions of interest.
3. **Region Features:** A tensor of shape $B \times C \times X \times Y$ giving features for each region; Each region is hence represented by an $X \times Y$ grid of C -dimensional features (in our experiments X, Y, C are (7,7,512) respectively as this is the spatial size of the last tensor in a VGGNet just before the first fully-connected layer).

Convolutional Anchors. Similar to Faster R-CNN [81], our localization layer predicts region proposal locations not in the raw absolute coordinate system of the image but by predicting offsets from a set of anchor boxes. In particular, we project each point in the $W' \times H'$ grid of input features back into the $W \times H$ image plane, and consider k anchor boxes of different aspect ratios centered at this projected point. For each of these k anchor boxes, the localization layer predicts a confidence score and four scalars regressing from the anchor to the predicted box coordinates. These are computed by passing the input feature map through a 3×3 convolution with 256 filters, a rectified linear nonlinearity, and a 1×1 convolution with $5k$ filters. This results in a tensor of shape $k \times W' \times H'$ containing scores and $4k \times W' \times H'$ offsets for all anchors.

Box Sampling. The previous step is done convolutionally (independently and in parallel across all spatial positions in the $W' \times H'$ grid), so the number of (scored) regions we produce at this stage of processing is $k \times W' \times H'$. As an example, with $k = 12$ and a typical image of size $W = 720, H = 540$ (i.e. $W' = 45, H' = 33$) the localization layer will at this stage predict $45 \times 33 \times 12 = 17,280$ scored regions. This is a very large number, so the localization layer will subsample the number of these regions to a much smaller number (i.e. $B \ll 17,280$ in this example). The subsampling (see ‘‘Sampling’’ in Figure 5.3) is done differently at training and at test time.

At training time, we follow the approach of [81] and sample a minibatch containing $B = 256$ boxes with at most $B/2$ positive regions and the rest negatives. A region is positive if it has an intersection over union (IoU) of at least 0.7 with some ground-truth region. In addition, the predicted region of maximal IoU with each ground-truth region is positive. A region is negative if it has IoU < 0.3 with all ground-truth regions. Our sampled minibatch contains $B_P \leq B/2$ positive regions and $B_N = B - B_P$ negative regions, sampled uniformly without replacement from the set of all

positive and all negative regions respectively. This subsampling is done only for efficiency since it is prohibitively expensive to process all regions for every image.

At test time we subsample using greedy non-maximum suppression (NMS) based on the predicted proposal confidences to select the $B = 300$ most confident proposals. Note that the use of NMS here is unsatisfying because our test time protocol differs from the training regime (where no NMS was carried out). Unfortunately it is non-trivial to overcome this limitation and designing a “neural” NMS or alternative differentiable suppression methods is a worthy goal for future work.

The coordinates and confidences of the sampled proposals are collected into tensors of shape $B \times 4$ and B respectively, and are output from the localization layer.

Box Regression. As an aside technical but important point, we adopt the parameterization of [30] to regress from anchors to the region proposals. Given an anchor box with center (x_a, y_a) , width w_a , and height h_a , our model predicts scalars (t_x, t_y, t_w, t_h) giving normalized offsets and log-space scaling transforms, so that the output region has center (x, y) and shape (w, h) given by

$$x = x_a + t_x w_a \qquad y = y_a + t_y h_a \qquad (5.1)$$

$$w = w_a \exp(t_w) \qquad h = h_a \exp(t_h) \qquad (5.2)$$

This parameterization has the appealing property that when the network predicts all zero then the predicted region will be exactly the anchor.

Bilinear Interpolation. After sampling, we are left with region proposals of varying sizes and aspect ratios. In order to interface with the full-connected recognition network and the RNN language model, we must extract a fixed-size feature representation for each variably sized region proposal.

To solve this problem, Fast R-CNN [30] proposes an RoI pooling layer where each region proposal is projected onto the $W' \times H'$ grid of convolutional features and divided into a coarse $X \times Y$ grid aligned to pixel boundaries by rounding. Features are max-pooled within each grid cell, resulting in an $X \times Y$ grid of output features.

The RoI pooling layer is a function of two inputs: convolutional features and region proposal coordinates. Gradients can be propagated backward from the output features to the input features, but not to the input proposal coordinates. To overcome this limitation, we replace the RoI pooling layer with with bilinear interpolation [32, 40], which is internally implemented by generating a sampling grid and using it in a bilinear sampler transformation (shown in Figure 5.3 at the end of the localization layer).

Concretely, given an input feature map U of shape $C \times W' \times H'$ and a region proposal, we interpolate the features of U to produce an output feature map V of shape $C \times X \times Y$. After projecting the region proposal onto U we follow [40] and compute a *sampling grid* G of shape $X \times Y \times 2$ associating each element of V with real-valued coordinates into U . If $G_{i,j} = (x_{i,j}, y_{i,j})$ then $V_{c,i,j}$ should be equal to U at $(c, x_{i,j}, y_{i,j})$; however since $(x_{i,j}, y_{i,j})$ are real-valued, we convolve

with a sampling kernel k and set

$$V_{c,i,j} = \sum_{i'=1}^W \sum_{j'=1}^H U_{c,i',j'} k(i' - x_{i,j}) k(j' - y_{i,j}). \quad (5.3)$$

We use bilinear sampling, corresponding to the kernel $k(d) = \max(0, 1 - |d|)$. The sampling grid is a linear function of the proposal coordinates, so gradients can be propagated backward into predicted region proposal coordinates. Running bilinear interpolation to extract features for all sampled regions gives a tensor of shape $B \times C \times X \times Y$, forming the final output from the localization layer.

Recognition Network

The recognition network is a fully-connected neural network that processes region features from the localization layer. The features from each region are flattened into a vector and passed through two full-connected layers, each using rectified linear units and regularized using Dropout. For each region this produces a code of dimension $D = 4096$ that compactly encodes its visual appearance. The codes for all positive regions are collected into a matrix of shape $B \times D$ and passed to the RNN language model.

In addition, we allow the recognition network one more chance to refine the confidence and position of each proposal region. It outputs a final scalar confidence of each proposed region and four scalars encoding a final spatial offset to be applied to the region proposal. These two outputs are computed as a linear transform from the D -dimensional code for each region. The final box regression uses the same parameterization as Section 5.2.1.

RNN Language Model

Following previous work [65, 43, 98, 18, 10] we use the region codes to condition an RNN language model [31, 68, 92]. Concretely, given a training sequence of tokens s_1, \dots, s_T , we feed the RNN $T+2$ word vectors $x_{-1}, x_0, x_1, \dots, x_T$, where $x_{-1} = \text{CNN}(I)$ is the region code encoded with a linear layer and followed by a ReLU non-linearity, x_0 corresponds to a special START token, and x_t encode each of the tokens s_t , $t = 1, \dots, T$. The RNN computes a sequence of hidden states h_t and output vectors y_t using a recurrence formula $h_t, y_t = f(h_{t-1}, x_t)$ (we use the LSTM [38] recurrence). The vectors y_t have size $|V| + 1$ where V is the token vocabulary, and where the additional one is for a special END token. The loss function on the vectors y_t is the average cross entropy, where the targets at times $t = 0, \dots, T-1$ are the token indices for s_{t+1} , and the target at $t = T$ is the END token. The vector y_{-1} is ignored. Our tokens and hidden layers have size 512.

At test time we feed the visual information x_{-1} to the RNN. At each time step we sample the most likely next token and feed it to the RNN in the next time step, repeating the process until the special END token is sampled.

5.2.2 Loss Function

During training our ground truth consists of positive boxes and descriptions. Our model predicts positions and confidences of sampled regions twice: in the localization layer and again in the recognition network. We use binary logistic losses for the confidences trained on sampled positive and negative regions. For box regression, we use a smooth L1 loss in transform coordinate space similar to [81]. The fifth term in our loss function is a cross-entropy term at every time-step of the language model.

We normalize all loss functions by the batch size and sequence length in the RNN. We searched over an effective setting of the weights between these contributions and found that a reasonable setting is to use a weight of 0.1 for the first four criteria, and a weight of 1.0 for captioning.

5.2.3 Training and Optimization

We train the full model end-to-end in a single step of optimization. We initialize the CNN with weights pretrained on ImageNet [84] and all other weights from a gaussian with standard deviation of 0.01. We use stochastic gradient descent with momentum 0.9 to train the weights of the convolutional network, and Adam [46] to train the other components of the model. We use a learning rate of 1×10^{-6} and set $\beta_1 = 0.9, \beta_2 = 0.99$. We begin fine-tuning the layers of the CNN after 1 epoch, and for efficiency we do not fine-tune the first four convolutional layers of the network.

Our training batches consist of a single image that has been resized so that the longer side has 720 pixels. Our implementation uses Torch7 [12]. One mini-batch runs in approximately 300ms on a Titan X GPU and it takes about three days of training for the model to converge.

5.3 Experiments

Dataset. Existing datasets that relate images and natural language either only include full image captions [9, 106], or ground words of image captions in regions but do not provide individual region captions [77]. We perform our experiments using the Visual Genome (VG) region captions dataset [50]. This dataset contains 94,313 images and 4,100,413 snippets of text (43.5 per image), each grounded to a region of an image. Images were taken from the intersection of MS COCO and YFCC100M [95], and annotations were collected on Amazon Mechanical Turk by asking workers to draw a bounding box on the image and describe its content in text. Example captions from the dataset include “cats play with toys hanging from a perch”, “newspapers are scattered across a table”, “woman pouring wine into a glass”, “mane of a zebra”, and “red light”.

Preprocessing. We collapse words that appear less than 15 times into a special <UNK> token, giving a vocabulary of 10,497 words. We strip referring phrases such as “there is...”, or “this seems to be a”. For efficiency we discard all annotations with more than 10 words (7% of annotations). We also

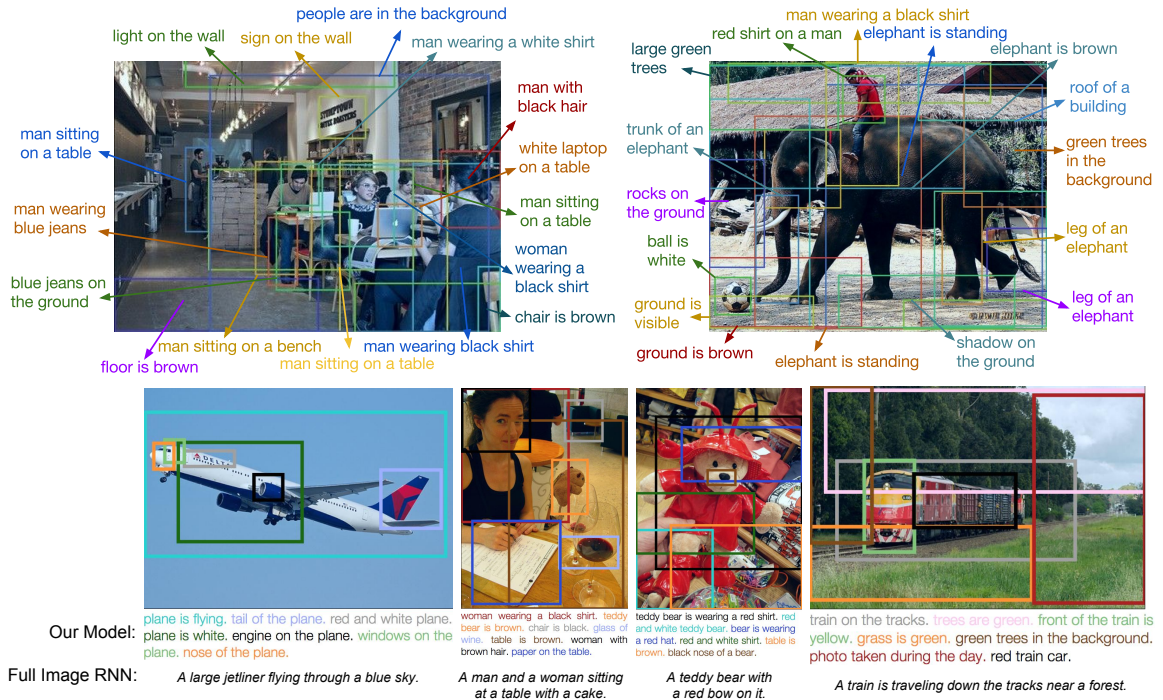


Figure 5.4: Example captions generated and localized by our model on test images. We render the top few most confident predictions. On the bottom row we additionally contrast the amount of information our model generates compared to the Full image RNN.

discard all images that have fewer than 20 or more than 50 annotations to reduce the variation in the number of regions per image. We are left with 87,398 images; we assign 5,000 each to val/test splits and the rest to train.

For test time evaluation we also preprocess the ground truth regions in the validation/test images by merging heavily overlapping boxes into single boxes with several reference captions. For each image we iteratively select the box with the highest number of overlapping boxes (based on IoU with threshold of 0.7), and merge these together (by taking the mean) into a single box with multiple reference captions. We then exclude this group and repeat the process.

5.3.1 Dense Captioning

In the dense captioning task the model receives a single image and produces a set of regions, each annotated with a confidence and a caption.

Evaluation metrics. Intuitively, we would like our model to produce both well-localized predictions (as in object detection) and accurate descriptions (as in image captioning).

Inspired by evaluation metrics in object detection [24, 62] and image captioning [97], we propose to measure the mean Average Precision (AP) across a range of thresholds for both localization and

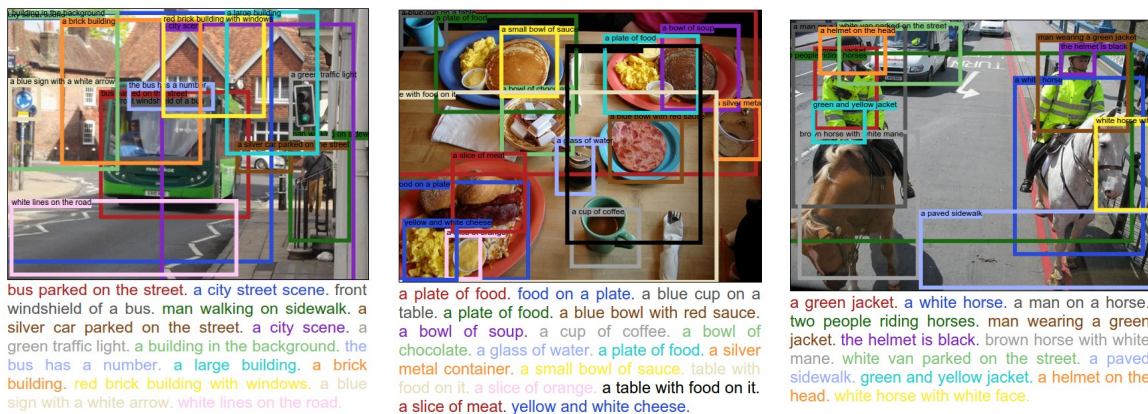


Figure 5.5: Additional predictions from the DenseCap model shown in a different format. The captions are shown both in the boxes but also listed below, in a decreasing order of their prediction confidence. The colors are arbitrary.

| Region source | Language (METEOR) | | | Dense captioning (AP) | | | Proposals | Test runtime (ms) | | |
|-----------------|-------------------|--------------|--------------|-----------------------|-------------|--------------|-------------|-------------------|-------------|--------------|
| | EB | RPN | GT | EB | RPN | GT | | CNN+Recog | RNN | Total |
| Image RNN [43] | 0.173 | 0.197 | 0.209 | 2.42 | 4.27 | <i>14.11</i> | 210ms | 2950ms | 10ms | 3170ms |
| Region RNN [43] | 0.221 | 0.244 | 0.272 | 1.07 | 4.26 | <i>21.90</i> | 210ms | 2950ms | 10ms | 3170ms |
| FCLN on EB [30] | 0.264 | 0.296 | 0.293 | 4.88 | 3.21 | <i>26.84</i> | 210ms | 140ms | 10ms | 360ms |
| Our model | 0.264 | 0.273 | 0.305 | 5.24 | 5.39 | <i>27.03</i> | 90ms | 140ms | 10ms | 240ms |

Table 5.1: Dense captioning evaluation on the test set of 5,000 images. The language metric is METEOR (high is good), our dense captioning metric is Average Precision (AP, high is good), and the test runtime performance for a 720×600 image with 300 proposals is given in milliseconds on a Titan X GPU (ms, low is good). EB, RPN, and GT correspond to EdgeBoxes [112], Region Proposal Network [81], and ground truth boxes respectively, **used at test time**. Therefore, the numbers in GT columns (italic) serve as upper bounds assuming perfect localization.

language accuracy. For localization we use intersection over union (IoU) thresholds .3, .4, .5, .6, .7. For language we use METEOR score thresholds 0, .05, .1, .15, .2, .25. We adopt METEOR since this metric was found to be most highly correlated with human judgments in settings with a low number of references [97]. We measure the average precision across all pairwise settings of these thresholds and report the mean AP.

To isolate the accuracy of language in the predicted captions without localization we also merge ground truth captions across each test image into a bag of references sentences and evaluate predicted captions with respect to these references without taking into account their spatial position.

Baseline models. Following Karpathy and Fei-Fei [43], we train only the Image Captioning model (excluding the localization layer) on individual, resized regions. We refer to this approach as a *Region RNN model*. To investigate the differences between captioning trained on full images or regions we also train the same model on full images and captions from MS COCO (*Full Image RNN model*).

At test time we consider three sources of region proposals. First, to establish an upper bound we evaluate the model on ground truth boxes (GT). Second, similar to [43] we use an external

region proposal method to extract 300 boxes for each test image. We use EdgeBoxes [112] (EB) due to their strong performance and speed. Finally, EdgeBoxes have been tuned to obtain high recall for objects, but our regions data contains a wide variety of annotations around groups of objects, stuff, etc. Therefore, as a third source of test time regions we follow Faster R-CNN [81] and train a separate Region Proposal Network (RPN) on the VG regions data. This corresponds to training our full model except without the RNN language model.

As the last baseline we reproduce the approach of Fast R-CNN [30], where the region proposals during training are fixed to EdgeBoxes instead of being predicted by the model (*FCLN on EB*). The results of this experiment can be found in Table 5.1. We now highlight the main takeaways.

Discrepancy between region and image level statistics. Focusing on the first two rows of Table 5.1, the Region RNN model obtains consistently stronger results on METEOR alone, supporting the difference in the language statistics present on the level of regions and images. Note that these models were trained on nearly the same images, but one on full image captions and the other on region captions. However, despite the differences in the language, the two models reach comparable performance on the final metric.

RPN outperforms external region proposals. In all cases we obtain performance improvements when using the RPN network instead of EB regions. The only exception is the FCLN model that was only trained on EB boxes. Our hypothesis is that this reflects people’s tendency of annotating regions more general than those containing objects. The RPN network can learn these distributions from the raw data, while the EdgeBoxes method was designed for high recall on objects. In particular, note that this also allows our model (FCLN) to outperform the FCLN on EB baseline, which is constrained to EdgeBoxes during training (5.24 vs. 4.88 and 5.39 vs. 3.21). This is despite the fact that their localization-independent language scores are comparable, which suggests that our model achieves improvements specifically due to better localization. Finally, the noticeable drop in performance of the FCLN on EB model when evaluating on RPN boxes (5.39 down to 3.21) also suggests that the EB boxes have particular visual statistics, and that the RPN boxes are likely out of sample for the FCLN on EB model.

Our model outperforms individual region description. Our final model performance is listed under the RPN column as 5.39 AP. In particular, note that in this one cell of Table 5.1 we report the performance of our full joint model instead of our model evaluated on the boxes from the independently trained RPN network. Our performance is quite a bit higher than that of the Region RNN model, even when the region model is evaluated on the RPN proposals (5.93 vs. 4.26). We attribute this improvement to the fact that our model can take advantage of visual information from the context outside of the test regions.

Qualitative results. We show example predictions of the dense captioning model in Figure 5.4 and Figure 5.5. The model generates rich snippet descriptions of regions and accurately grounds the captions in the images. For instance, note that several parts of the elephant in Figure 5.4 are

| | Ranking | | | | Localization | | | |
|--------------------------|-------------|-------------|-------------|-----------|--------------|--------------|--------------|--------------|
| | R@1 | R@5 | R@10 | Med. rank | IoU@0.1 | IoU@0.3 | IoU@0.5 | Med. IoU |
| Full Image RNN [43] | 0.10 | 0.30 | 0.43 | 13 | - | - | - | - |
| EB + Full Image RNN [43] | 0.11 | 0.40 | 0.55 | 9 | 0.348 | 0.156 | 0.053 | 0.020 |
| Region RNN [30] | 0.18 | 0.43 | 0.59 | 7 | 0.460 | 0.273 | 0.108 | 0.077 |
| Our model (FCLN) | 0.27 | 0.53 | 0.67 | 5 | 0.560 | 0.345 | 0.153 | 0.137 |

Table 5.2: Results for image retrieval experiments. We evaluate ranking using recall at k ($R@K$, higher is better) and median rank of the target image (Med.rank, lower is better). We evaluate localization using ground-truth region recall at different IoU thresholds ($IoU@t$, higher is better) and median IoU (Med. IoU, higher is better). Our method outperforms baselines at both ranking and localization.

correctly grounded and described (“trunk of an elephant”, “elephant is standing”, and both “leg of an elephant”). The same is true for the airplane example, where the tail, engine, nose and windows are correctly localized. Common failure cases include repeated detections (e.g. the elephant is described as standing twice).

Runtime evaluation. Our model is efficient at test time: a 720×600 image is processed in 240ms. This includes running the CNN, computing $B = 300$ region proposals, and sampling from the language model for each region.

Table 5.1 (right) compares the test-time runtime performance of our model with baselines that rely on EdgeBoxes. Regions RNN is slowest since it processes each region with an independent forward pass of the CNN; with a runtime of 3170ms it is more than $13\times$ slower than our method.

FCLN on EB extracts features for all regions after a single forward pass of the CNN. Its runtime is dominated by EdgeBoxes, and it is $\approx 1.5\times$ slower than our method.

Our method takes 88ms to compute region proposals, of which nearly 80ms is spent running NMS to subsample regions in the Localization Layer. This time can be drastically reduced by using fewer proposals: using 100 region proposals reduces our total runtime to 166ms.

5.3.2 Image Retrieval using Regions and Captions

In addition to generating novel descriptions, our dense captioning model can support image retrieval using natural-language queries, and can localize these queries in retrieved images. We evaluate our model’s ability to correctly retrieve images and accurately localize textual queries.

Experiment setup. We use 1000 random images from the VG test set for this experiment. We generate 100 test queries by repeatedly sampling four random captions from some image and then expect the model to correct retrieve the source image for each query.

Evaluation. To evaluate ranking, we report the fraction of queries for which the correct source image appears in the top k positions for $k \in \{1, 5, 10\}$ (recall at k) and the median rank of the correct image across all queries.

To evaluate localization, for each query caption we examine the image and ground-truth bounding box from which the caption was sampled. We compute IoU between this ground-truth box and the

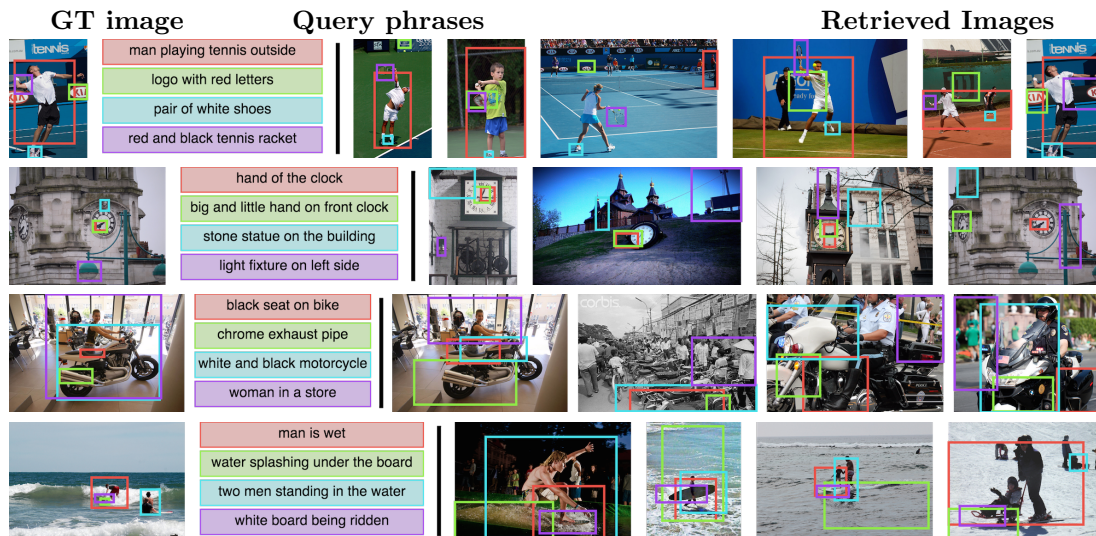


Figure 5.6: Example image retrieval results using our dense captioning model. From left to right, each row shows a ground-truth test image, ground-truth region captions describing the image, and the top images retrieved by our model using the text of the captions as a query. Our model is able to correctly retrieve and localize people, animals, and parts of both natural and man-made objects.

model’s predicted grounding for the caption. We then report the fraction of query caption for which this overlap is greater than a threshold t for $t \in \{0.1, 0.3, 0.5\}$ (recall at t) and the median IoU across all query captions.

Models. We compare the ranking and localization performance of full model with baseline models from Section 5.3.1.

For the Full Image RNN model trained on MS COCO, we compute the probability of generating each query caption from the entire image and rank test images by mean probability across query captions. Since this does not localize captions we only evaluate its ranking performance.

The Full Image RNN and Region RNN methods are trained on full MS COCO images and ground-truth VG regions respectively. In either case, for each query and test image we generate 100 region proposals using EdgeBoxes and for each query caption and region proposal we compute the probability of generating the query caption from the region. Query captions are aligned to the proposal of maximal probability, and images are ranked by the mean probability of aligned caption / region pairs.

The process for the full FCLN model is similar, but uses the top 100 proposals from the localization layer rather than EdgeBoxes proposals.

Discussion. Figure 5.6 shows examples of ground-truth images, query phrases describing those images, and images retrieved from these queries using our model. Our model is able to localize small objects (“hand of the clock”, “logo with red letters”), object parts, (“black seat on bike”, “chrome

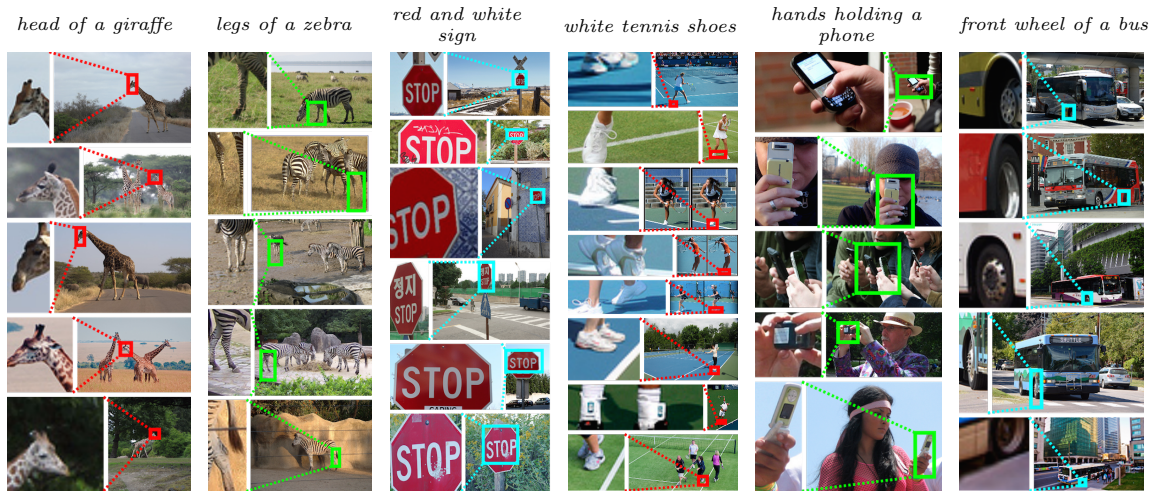


Figure 5.7: Example results for open world detection. We use our dense captioning model to localize arbitrary pieces of text in images, and display the top detections on the test set for several queries.

exhaust pipe”), people (“man is wet”) and some actions (“man playing tennis outside”).

Quantitative results comparing our model against the baseline methods is shown in Table 5.2. The relatively poor performance of the Full Image RNN model (Med. rank 13 vs. 9,7,5) may be due to mismatched statistics between its train and test distributions: the model was trained on full images, but in this experiment it must match region-level captions to whole images (Full Image RNN) or process image regions rather than full images (EB + Full Image RNN).

The Region RNN model does not suffer from a mismatch between train and test data, and outperforms the Full Image RNN model on both ranking and localization. Compared to Full Image RNN, it reduces the median rank from 9 to 7 and improves localization recall at 0.5 IoU from 0.053 to 0.108.

Our model outperforms the Region RNN baseline for both ranking and localization under all metrics, further reducing the median rank from 7 to 5 and increasing localization recall at 0.5 IoU from 0.108 to 0.153.

The baseline uses EdgeBoxes which was tuned to localize objects, but not all query phrases refer to objects. Our model achieves superior results since it learns to propose regions from the training data.

Open-world Object Detection Using the retrieval setup described above, our dense captioning model can also be used to localize arbitrary pieces of text in images. This enables “open-world” object detection, where instead of committing to a fixed set of object classes at training time we can specify object classes using natural language at test-time. We show example results for this task in Figure 5.7, where we display the top detections on the test set for several phrases.

Our model can detect animal parts (“head of a giraffe”, “legs of a zebra”) and also understands

some object attributes (“red and white sign”, “white tennis shoes”) and interactions between objects (“hands holding a phone”). The phrase “front wheel of a bus” is a failure case: the model correctly identifies wheels of buses, but cannot distinguish between the front and back wheel.

5.4 Conclusions

We introduced the dense captioning task, which requires a model to simultaneously localize and describe regions of an image. To address this task we developed the FCLN architecture, which supports end-to-end training and efficient test-time performance. Our FCLN architecture is based on recent CNN-RNN models developed for image captioning but includes a novel, differentiable localization layer that can be inserted into any neural network to enable spatially-localized predictions. Our experiments in both generation and retrieval settings demonstrate the power and efficiency of our model with respect to baselines related to previous work, and qualitative experiments show visually pleasing results. In future work we would like to relax the assumption of rectangular proposal regions (which is in principle a trivial extension due to our use of spatial transformers in the localization layer) and to discard test-time non-maximum suppression in favor of a trainable spatial suppression layer.

6

Conclusions

We’ve witnessed rapid advances in the field of Computer Vision over the last few years. An often cited benchmark that exemplifies this progress is the ImageNet Large-Scale Visual Recognition Challenge, where we’ve seen the top-5 error rate for image classification (a core visual recognition problem) decrease from 27% in 2010 down to 3% in 2016. To put this in context, this performance rivals human recognition capabilities, or even surpasses it in some fine-grained categories such as recognizing different breeds of dogs [84].

However, far from diminishing opportunities for research in Computer Vision, progress on this core task has energized the field and unlocked a wide variety of ambitious problems that once defied our attempts but suddenly seem within our grasp. In particular, in this dissertation we developed models and techniques that push the frontier of visual recognition by expanding the label space from a discrete set of categories to the space of natural language utterances. We argued that doing so is 1) a stepping stone towards Artificial Intelligence agents that can perceive the visual world and interact with us in natural language (in a manner similar to that imagined by Turing at the conception of the field), 2) a yet unsolved and critical next frontier in Computer Vision and 3) desirable from a practical perspective since humans communicate and serialize knowledge in this encoding format.

Concretely, in Chapter 3 we developed models that can match images and sentences. This allowed us to annotate images with sentence descriptions and, conversely, retrieve images from a database for any text query. Our final model not only allowed us to perform this task to high accuracy, but allowed us to visually inspect the model’s predictions (e.g. seeing exactly which parts of the image caused a model to think that it might depict “a man playing accordion in a public area”).

In Chapter 4 we introduced a model that can take an image and generate a new sentence description without being constrained to merely choosing from a finite collection of sentences written by humans. The model generates qualitatively sensible sentences that are novel approximately 50% of the time, and we showed in quantitative evaluation that the model outperforms previous captioning methods based on a ranking approach.

Finally, in Chapter 5 we expanded the image captioning model to both detect and describe all salient aspects of an image. The final model can process an image in as little as a few hundred milliseconds and covers images with detailed grounded descriptions at all scales, from holistic image-level scale as in image captioning to the level of individual object or animal parts.

From a modeling perspective, the models we developed fall under the category of deep learning approaches, where each model defines a single differentiable function from raw inputs to raw outputs. The function's parameters are trained by optimizing the final objective of interest end-to-end on data. This approach offers many practical benefits: it relies on relatively simple and homogeneous operations repeated in layers which decreases code complexity, it uses parallel computations that can be carried out very efficiently on specialized Graphics Processing Unit cards (GPUs), and all components of the model share the same objective, which in practice often results in high performance. We hope that many of the architectural elements featured in our models can be reused and help inspire future work.

Despite recent rapid progress in visual recognition, it is clear that many challenges still remain before we can realize Turing's vision of machines that can sense the visual world and interact with us through natural language. The remaining challenges are best illustrated with a concrete example. Consider the image in Figure 6.1:



Figure 6.1: An image depicting an amusing situation.

It only takes a few short moments to go from glancing at the image to a feeling of amusement, in a realization of the full depicted situation. The steps we take to fully perceive this visual scene are numerous and astonishing. It is instructive to enumerate at least some of them:

- Starting off with some of the easier concepts, you recognize that the image contains several people in suits standing in a hallway.

- You recognize that there are 3 mirrors in the scene so some of those people are “fake” replicas from different viewpoints.
- You probably recognize one of these people as a specific individual (President Obama) from only the few pixels that make up his face and perhaps some of the wider context that includes his clothes and the other people.
- You recognize that there’s a person standing on a scale, even though the scale occupies only very few white pixels that blend in with the white walls. However, the body pose of the person as well as his eye gaze possibly help disambiguate this fact.
- You recognize that Obama has his foot positioned just slightly on top of the scale. In particular, you reason about the 3D structure of the scene rather than simply the 2D coordinate system of the image.
- You’re also aware of basic physics: Obama’s center of mass is slightly ahead of him. Therefore he is leaning in on the scale, applying some force on it. In this inference you’re taking advantage of intuitive physics.
- You understand that scales measure force that is applied vertically on them, so it will over-estimate the weight of the person standing on it. That is, you’re reasoning about object affordances, their function and purpose.
- The person measuring his weight is not aware of Obama behind him, which you can derive based on an understanding of the field of view of a person, and that the slight push from the foot is unlikely to be felt by him.
- You understand that people are self-conscious about their weight. You also understand that he is reading off the scale measurement, and that shortly the over-estimated weight might confuse him because it will likely be much higher than what he expects. Notice that you’re reasoning about the state of mind of people and the implications of events that are about to unfold seconds after this image was taken. You also reason about what information is likely available to different people. Collectively, you’ve already made a large number of inferences based on intuitive psychology.
- There are several people in the back who find the person’s imminent confusion amusing. That is, you reason about the state of mind of people, their view of the state of mind of another person, and the dynamics of these states over time.
- Finally, the fact that the perpetrator of the joke is the president further contributes to the uniqueness of the situation. That is, you’re bringing knowledge of social roles and what actions are more or less likely to be undertaken by different people based on their status and identity.

DenseCap is an arguably large step forward in comparison to labeling an image with a category, but its performance on this particular image would clearly be abysmal. The model might correctly recognize hundreds of concepts in the image: it might identify and describe all of the people, their suits, the checkered floor, the lockers, lights on the ceiling, or possibly even distinguish the mirrors. And yet, it would completely miss the point of the scene, barely scratching at the surface of the semantic content of the depicted situation. In this light, DenseCap is a texture-based sequence memorizer with some limited generalization, not a probabilistic inference engine that pieces together an understanding of a complex scene from both the visual evidence and a rich general understanding of how the world works: that people have minds, goals, beliefs, identities and social status, or that objects have specific affordances, intended uses, or that they obey certain physical dynamics.

How can we endow computers with an understanding of so many interconnected abstract concepts and knowledge?

First, it is important to recognize that a necessary (but not sufficient) condition is that the information about the world must be made available to the computer. This already presents many practical difficulties related to data collection and storage. For instance, in this concrete example, I recognize Obama from news articles and TV, and a computer cannot do so until it somehow gains access to the same data. As a more problematic example, I understand how the mechanical scale in image works because I've interacted with one myself in the real world: I stood on it and saw its reading change, I played with the setting of its counter weights, I shifted my weight around and saw it react. Therefore, an argument can be made that I've benefited a great deal from embodied interaction with the environment and my ability to run small experiments that disambiguate between different likely hypotheses of world dynamics. This line of thought argues that we might not reach the same level of understanding in computers until they can also interact with the world in the same way we have done for many years of our upbringing.

Second, it is insightful to note that the representations of these abstract relationships are difficult to manually encode in some formal language and provide to the computer directly under the umbrella of supervised learning. For instance, CYC [59] is a popular example of an attempt to assemble a comprehensive ontology and knowledge base of common knowledge in a formal language, which turned out to be very challenging. Instead, it seems that a more promising approach is to allow a model to discover its own internal representations, similar to word embedding methods where the structure and relationships emerge as a result of optimizing some objective.

Lastly, one central challenge lies in how we can design architectures that can model abstract concepts and theories (e.g. that people have a finite field of view, or that scales measure weight) and how they can be acquired, stored and manipulated in an end-to-end learning framework. Furthermore, one would like to represent distributions over theories and come up with objectives that encourage agents to disambiguate between competing hypotheses (e.g. figuring out how a scale works).

To make further progress, it is these kinds of problems that we must turn to next.

Bibliography

- [1] Andrei Barbu, Alexander Bridge, Zachary Burchill, Dan Coroian, Sven Dickinson, Sanja Fidler, Aaron Michaux, Sam Mussman, Siddharth Narayanaswamy, Dhaval Salvi, et al. Video in sentences out. In *UAI 2012*, 2012.
- [2] Kobus Barnard, Pinar Duygulu, David Forsyth, Nando De Freitas, David M Blei, and Michael I Jordan. Matching words and pictures. *The Journal of Machine Learning Research*, 2003.
- [3] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.
- [4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.
- [5] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Frédéric Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*. Springer, 2006.
- [6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.
- [7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- [8] Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *Association for Computational Linguistics*, 2016.
- [9] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.
- [10] Xinlei Chen and C Lawrence Zitnick. Learning a recurrent visual representation for image caption generation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

- [11] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17(11):1875–1886, 2015.
- [12] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, Advances in Neural Information Processing Systems Workshop*, 2011.
- [13] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of Language Resources and Evaluation Conference*, volume 6, pages 449–454, 2006.
- [14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2009.
- [15] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Association for Computational Linguistics 2014 Workshop on Statistical Machine Translation*, 2014.
- [16] Jacob Devlin, Saurabh Gupta, Ross Girshick, Margaret Mitchell, and C Lawrence Zitnick. Exploring nearest neighbor approaches for image captioning. *arXiv preprint arXiv:1505.04467*, 2015.
- [17] 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*, pages 647–655, 2014.
- [18] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.
- [19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [20] Pinar Duygulu, Kobus Barnard, Joao FG de Freitas, and David A Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *European conference on computer vision*, pages 97–112. Springer, 2002.
- [21] Desmond Elliott and Frank Keller. Image description using visual dependency representations. In *Proceedings of the Empirical Methods on Natural Language Processing*, pages 1292–1302, 2013.

- [22] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [23] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014.
- [24] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.
- [25] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. From captions to visual concepts and back. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1473–1482, 2015.
- [26] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In *Proceedings of the European Conference on Computer Vision*, pages 15–29. Springer, 2010.
- [27] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, 2013.
- [28] Kunihiro Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.
- [29] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014.
- [30] Ross Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision*, 2015.
- [31] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- [32] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *Proceedings of the International Conference on Machine Learning*, 2015.

- [33] Abhinav Gupta and Larry S Davis. Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers. In *European conference on computer vision*, pages 16–29. Springer, 2008.
- [34] Abhinav Gupta, Praveen Srinivasan, Jianbo Shi, and Larry S Davis. Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2012–2019. IEEE, 2009.
- [35] Ankush Gupta and Prashanth Mannem. From image annotation to image description. In *Neural information processing*. Springer, 2012.
- [36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proceedings of the European Conference on Computer Vision*, pages 346–361. Springer, 2014.
- [37] Geoffrey E Hinton, James L McClelland, and David E Rumelhart. Distributed representations, parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, 1986.
- [38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [39] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 2013.
- [40] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *Advances in Neural Information Processing Systems*, 2015.
- [41] Yangqing Jia, Mathieu Salzmann, and Trevor Darrell. Learning cross-modality similarity for multinomial data. In *IEEE International Conference on Computer Vision*, 2011.
- [42] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. Denscap: Fully convolutional localization networks for dense captioning. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016.
- [43] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014.
- [44] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *International Conference on Learning Representations Workshop*, 2016.

- [45] Andrej Karpathy, Armand Joulin, and Li Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems*, 2014.
- [46] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*, 2015.
- [47] Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *Transactions of the Association for Computational Linguistics*, 2015.
- [48] Ryan Kiros, Richard S Zemel, and Ruslan Salakhutdinov. Multimodal neural language models. *Proceedings of the International Conference on Machine Learning*, 2014.
- [49] Chen Kong, Dahua Lin, Mohit Bansal, Raquel Urtasun, and Sanja Fidler. What are you talking about? text-to-image coreference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014.
- [50] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016.
- [51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.
- [52] Girish Kulkarni, Visruth Premraj, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C Berg, and Tamara L Berg. Baby talk: Understanding and generating simple image descriptions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2011.
- [53] Polina Kuznetsova, Vicente Ordonez, Alexander C. Berg, Tamara L. Berg, and Yejin Choi. Collective generation of natural image descriptions. In *Association for Computational Linguistics*, 2012.
- [54] Polina Kuznetsova, Vicente Ordonez, Alexander C Berg, Tamara L Berg, and Yejin Choi. Generalizing image captions for image-text parallel corpus. In *Association for Computational Linguistics*, 2013.
- [55] Polina Kuznetsova, Vicente Ordonez, Tamara L Berg, UNC Chapel Hill, and Yejin Choi. Treetalk: Composition and compression of trees for image descriptions. *Transactions of the Association for Computational Linguistics*, 2(10):351–362, 2014.
- [56] Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

- [57] Y LeCun, B Boser, JS Denker, D Henderson, RE Howard, W Hubbard, and LD Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems 2, NIPS 1989*, pages 396–404. Morgan Kaufmann Publishers, 1990.
- [58] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [59] Douglas B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
- [60] Siming Li, Girish Kulkarni, Tamara L Berg, Alexander C Berg, and Yejin Choi. Composing simple image descriptions using web-scale n-grams. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 220–228. Association for Computational Linguistics, 2011.
- [61] Dahua Lin, Sanja Fidler, Chen Kong, and Raquel Urtasun. Visual semantic search: Retrieving videos via complex textual queries. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2657–2664. IEEE, 2014.
- [62] 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 *Proceedings of the European Conference on Computer Vision*, pages 740–755. Springer, 2014.
- [63] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2015.
- [64] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.
- [65] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L Yuille. Explain images with multi-modal recurrent neural networks. *arXiv preprint arXiv:1410.1090*, 2014.
- [66] Cynthia Matuszek*, Nicholas FitzGerald*, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. A Joint Model of Language and Perception for Grounded Attribute Learning. In *Proceedings of the 2012 International Conference on Machine Learning*, Edinburgh, Scotland, June 2012.
- [67] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [68] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

- [69] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013.
- [70] Margaret Mitchell, Xufeng Han, Jesse Dodge, Alyssa Mensch, Amit Goyal, Alex Berg, Kota Yamaguchi, Tamara Berg, Karl Stratos, and Hal Daumé III. Midge: Generating image descriptions from computer vision detections. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 747–756. Association for Computational Linguistics, 2012.
- [71] Yasuhide Mori, Hironobu Takahashi, and Ryuichi Oka. Image-to-word transformation based on dividing and vector quantizing images with words. In *First International Workshop on Multimedia Intelligent Storage and Retrieval Management*, pages 1–9. Citeseer, 1999.
- [72] Vicente Ordonez, Xufeng Han, Polina Kuznetsova, Girish Kulkarni, Margaret Mitchell, Kota Yamaguchi, Karl Stratos, Amit Goyal, Jesse Dodge, Alyssa Mensch, et al. Large scale retrieval and generation of image descriptions. *International Journal of Computer Vision*, 2015.
- [73] Vicente Ordonez, Girish Kulkarni, and Tamara L Berg. Im2text: Describing images using 1 million captioned photographs. In *Advances in Neural Information Processing Systems*, 2011.
- [74] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.
- [75] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the International Conference on Machine Learning*, 2013.
- [76] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the Empirical Methods on Natural Language Processing*, 2014.
- [77] Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2641–2649, 2015.
- [78] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.
- [79] Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. Collecting image annotations using amazon’s mechanical turk. In *Proceedings of the Association for Computational Linguistics HLT 2010 Workshop on Creating Speech and Language Data with Amazon’s Mechanical Turk*, pages 139–147. Association for Computational Linguistics, 2010.

- [80] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016.
- [81] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, 2015.
- [82] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [83] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
- [84] 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. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [85] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 1997.
- [86] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2014.
- [87] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.
- [88] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Proceedings of the International Conference on Learning Representations*, 2015.
- [89] Richard Socher and Li Fei-Fei. Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2010.
- [90] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Association for Computational Linguistics*, 2014.

- [91] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the International Conference on Machine Learning*, 2011.
- [92] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2011.
- [93] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [94] Christian Szegedy, Scott Reed, Dumitru Erhan, and Dragomir Anguelov. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014.
- [95] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.
- [96] T Tieleman and Geoffrey E Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude., 2012.
- [97] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4566–4575, 2015.
- [98] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.
- [99] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.
- [100] Bernard Widrow, Marcian E Hoff, et al. Adaptive switching circuits. In *IRE WESCON convention record*, volume 4, pages 96–104. New York, 1960.
- [101] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On multiplicative integration with recurrent neural networks. *arXiv preprint arXiv:1606.06630*, 2016.
- [102] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *Proceedings of the International Conference on Machine Learning*, 2015.

- [103] Yezhou Yang, Ching Lik Teo, Hal Daumé III, and Yiannis Aloimonos. Corpus-guided sentence generation of natural images. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 444–454. Association for Computational Linguistics, 2011.
- [104] Benjamin Z Yao, Xiong Yang, Liang Lin, Mun Wai Lee, and Song-Chun Zhu. I2t: Image parsing to text description. *Proceedings of the IEEE*, 98(8):1485–1508, 2010.
- [105] Mark Yatskar, Lucy Vanderwende, and Luke Zettlemoyer. See no evil, say no evil: Description generation from densely labeled images. *Lexical and Computational Semantics*, 2014.
- [106] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Association for Computational Linguistics*, 2014.
- [107] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.
- [108] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.
- [109] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 818–833. Springer, 2014.
- [110] Weixiong Zhang. *State-space search: Algorithms, complexity, extensions, and applications*. Springer Science & Business Media, 1999.
- [111] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 19–27, 2015.
- [112] C. Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *Proceedings of the European Conference on Computer Vision*, 2014.
- [113] C Lawrence Zitnick, Devi Parikh, and Lucy Vanderwende. Learning the visual interpretation of sentences. *Proceedings of the International Conference on Computer Vision*, 2013.