Vision by Alignment

by

Adam Davis Kraft

B.S., Massachusetts Institute of Technology (2005)
M.Eng., Massachusetts Institute of Technology (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2018

Author . . . . . .
Department of Electrical Engineering and Computer Science
January 18, 2018

Certified by . . . . . .
Patrick H. Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair of the Department Committee on Graduate Students

Vision by Alignment

by

Adam Davis Kraft

Submitted to the Department of Electrical Engineering and Computer Science
on January 18, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Human visual intelligence is robust. Vision is versatile in its variety of tasks and operating conditions, it is flexible, adapting facilely to new tasks, and it is introspective, providing compositional explanations for its findings. Vision is fundamentally underdetermined, but it exists in a world that abounds with constraints and regularities perceived not only through vision but through other senses as well.

These observations suggest that the imperative of vision is to exploit all sources of information to resolve ambiguity. I propose an alignment model for vision, in which computational specialists eagerly share state with their neighbors during ongoing computations, availing themselves of neighbors' partial results in order to fill gaps in evolving descriptions. Connections between specialists extend across sensory modalities, so that the computational machinery of many senses may be brought to bear on problems with strictly-visual inputs.

I anticipate that this alignment process accounts for vision's robust attributes, and I call this prediction the **alignment hypothesis**. In this document I lay the groundwork for evaluating the hypothesis. I then demonstrate progress toward that goal, by way of the following contributions:

- I performed an experiment to investigate and characterize the ways that high-performing computer-vision models fall short of robust perception, and evaluated whether alignment models can address the shortcomings. The experiment, which relied on a procedure to remove signal energy from natural images while preserving high classification confidence by a neural network, revealed that the type of object depicted in the original image is a strong predictor of whether humans recognize the reduced-energy image.

- I implemented an alignment model based on a network of propagators. The model can use constraints to infer locations and heights of pedestrians and locations of occluding objects in an outdoor urban scene. I used the results of the effort to refine the requirements of mechanisms to use in building alignment models.

- I implemented an alignment model based on neural networks. Alignment-motivated design empowers the model, trained to estimate depth maps from single images, to perform the additional task of depth super-resolution without retraining. The design thus demonstrates flexibility, a property of robust vision systems.

Thesis Supervisor: Patrick H. Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Acknowledgments

Foremost I thank my advisor, Patrick Winston. My meandering path through AI research has given me the impression that AI progress will improve dramatically when more people adopt a few of Patrick's habits and ideas. There is no substitute for learning Patrick's habits and ideas from Patrick. If you do not have that opportunity, though, then read the rest of this paragraph because it will make you smarter. To understand human intelligence, you must reject all meager proxies for it. Seek a principled approach, reject mechanistic approaches, and do not compromise. Tell yourself the right story. Explain everything as simply and clearly as possible. Have a vision that is worth dedicating an entire career to, and solve problems that make progress toward your vision. When you tell people about the problems that you solve, make sure you first tell them what your vision is. Tell them in a way that will inspire somebody to dedicate an entire career to it.

Gerald Sussman and Shimon Ullman served on my thesis committee. Gerry's thoughtful critique of this document was invaluable. I have come to appreciate that an hour-long conversation with Gerry easily imparts weeks of ideas to revisit, and I'm grateful to have had the opportunity to work with him. Similarly, Shimon's work has always inspired me, and has helped me to understand which problems in vision are essential to solve.

I am grateful to all of my friends and colleagues at MIT CSAIL and especially to Dylan Holmes and Michael Fleder. I thank Dylan for patient critique, for countless conversations about this work, and for sharing a bright outlook on a future with robust AI. Dylan generously applied his expertise to design several of the best illustrations in this document. I have been very fortunate to benefit from Michael's deep understanding of machine-learning concepts and from his talent for clear explanation.

I owe a debt of gratitude to my friends who read this document at various stages of completion and provided me with feedback: to Avril Kenney for her extremely thorough and perceptive attention to detail and for keeping me honest; to Blake Stacey, whose science writing expertise led to enormously helpful suggestions; and to Brian Neltner, for holding me accountable to high standards. Additionally, Robert McIntyre generously supplied me with a working set of custom tools and templates for compiling this document.

Michael Coen, Jeffrey Siskind, Gadi Geiger, and Sajit Rao provided inspiration for this work, particularly in its early stages. Their work continues to inspire me. Gadi's weekly research meeting was an invaluable resource, early on in this work.

Generous support through DARPA and NSF made this work possible. I am especially grateful to James Donlon for his vision and leadership in both the Mind's Eye and Robust Intelligence programs. This work was sponsored by awards FA8750-05-2-0274, D11AP00250, W911NF-10-2-0065, and IIS-1421065.

Contents

List of Figures

1 Vision

As the remnants of AI winter thaw rapidly, excitement over machine learning's rapid pace of achievement is palpable. Machines now outperform humans on tasks such as object-detection, a milestone that was far beyond reach less than a decade ago. Perhaps the greatest opportunity in AI research, though, is the opportunity to gain a deep computational understanding of the way people think. Despite outstanding technical achievements, progress on this front has been slow. The disparity between performance and understanding is especially striking in computer vision.

To understand human visual intelligence is to account for its astonishing versatility, its flexibility, and the coherence and depth of the explanations it offers for its observations. A closer look at the state of the art in computer vision reveals that it has not yet achieved any these abilities, despite the tremendous technological value of its achievements to date.

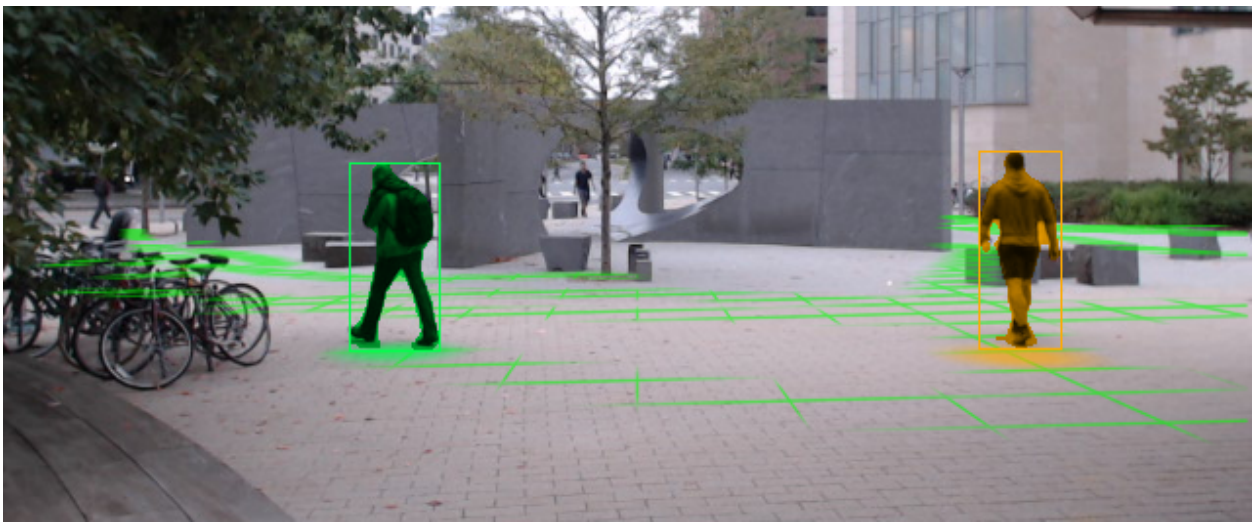
This is the story of my first steps toward describing, implementing, and understanding robust visual intelligence. The story revolves around the **alignment hypothesis** that you learn about in Section 1.1. The highlights and major developments in the story are depicted in Figure 1.

In Chapter 2, you will find out about an experiment in which I investigated and characterized discrepancies between a neural network's visual ability and human visual intelligence. The implications of the experiment reinforce the need for robust vision architectures. As part of the experiment, I developed a procedure to isolate the features of images that support confident classification by a neural network, and asked whether those features also support classification by humans. A surprising outcome of the experiment is that whether or not the neural network features are human-recognizable depends primarily on the type of object depicted in the image, rather than on other details of the image.

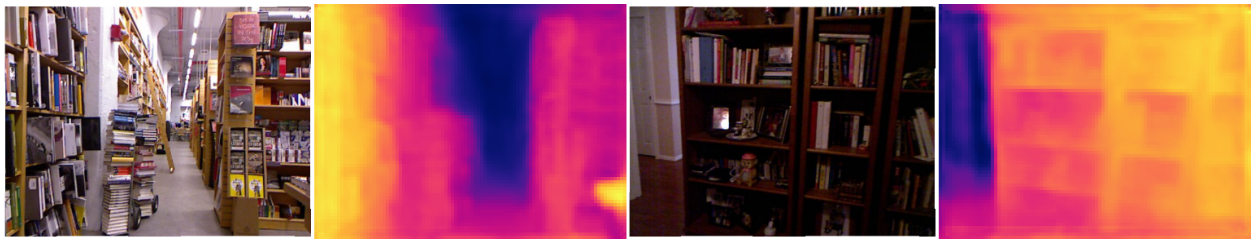
Then, in Chapters 3 and 4, you will learn about my work in implementing alignment-driven vision systems using a propagator architecture. A significant outcome of that work is a vision system that tracks people as they move, and uses the tracking along with constraints like *people must be supported in order to walk* to infer geometric properties of the scene and the actors.

In Chapter 5 you will see my first steps toward applying alignment principles to the design of neural networks. A significant result is that the design empowers a neural network to accomplish the task of depth super-resolution despite that it was trained just for the task of depth estimation.

(a) Images and minimal features



(b) Pedestrians and scene geometry



(c) Images and generated depth maps

Figure 1: Visual summary of highlights and developments
Inputs followed by outputs, shown in (a), of a signal-energy reduction algorithm that preserves neural network classification confidence. A propagator system, with output shown in (b), uses the tracks of pedestrians combined with knowledge about average human size to help identify the ground plane and refine heights of pedestrians. A neural network performs both depth estimation from single images, and depth up-sampling, with inputs and outputs shown in (c).

1.1 Motivation

Human visual intelligence combines a remarkable set of properties. Vision is **versatile** in its capability to perform many important tasks, often simultaneously and under widely varying conditions. Vision is **flexible**, able to adapt to new tasks quickly and with little preparation, including tasks such as driving that are very different from those most influential over our evolutionary development. Vision is **introspective**, providing us with mutually reinforcing ways to corroborate, understand and explain its findings: objects have discernible parts and properties, events have causes and effects, and actors have intent.

How can we capture natural vision's versatility, flexibility, and introspection in a computational model? How can we move on to engineer AI systems that have these properties, so that we may interact with the systems in our natural environment and on our terms? To take steps toward answering these questions, I observe four factors that most prominently constrain and guide the development of natural vision systems:

1. The process of forming a 2D image from a natural world state is not invertible. Further complicating matters, the signal that eyes receive from the world is fragmentary and ambiguous. Signal noise, occlusion, and other environmental conditions guarantee that, in all cases of interest, there is no simple heuristic to reconstruct the generating world state from an image.

2. Our many sensory modes collect information about percepts simultaneously. For example, an infant's hand making contact with an object causes the infant to perceive impact via vision, hearing, touch, and proprioceptive sensory modes.

3. Constraints and regularities in the world create abundant contextual cues for vision. Certain objects are more likely than others to move, be composed of specific parts and materials, or participate in common configurations with other objects. Physical constraints force objects to be supported, and, for example, very distant objects to appear blue from atmospheric color distortion.

4. Throughout evolution, our need to understand visual stimuli has been directed by goals. Necessities of life such as finding food and avoiding danger ruthlessly optimized the visual intelligence of our recent ancestors for those goals, without regard for any notion of ground truth independent of those goals. The architectural results of that optimization provide the foundation for human visual intelligence.

Each observation has strong implications for natural vision systems. Vision is fundamentally hard because the signal our eyes receive is ambiguous. A path toward resolving such ambiguity exists only because sensory modes collect information simultaneously, and because constraints and regularities abound in nature. Because vision developed to serve survival goals, whatever mechanisms evolved to resolve visual ambiguity must do so aggressively, exploiting all available sources of information with minimal delay. The four observations together suggest that the computational imperative (Winston and Holmes [2018]) of natural vision systems is to exploit all sources of information to resolve ambiguity in a goal-directed framework. This computational imperative necessarily blurs the boundaries between vision systems and other perceptual machinery.

Such intuition stimulates what I call the **alignment hypothesis: robust perception requires pervasive alignment of partial information throughout a multimodal network.** In this statement of the hypothesis, **robust perception** refers to perception that achieves the versatile, flexible, and introspective attributes of natural systems. Despite recent sweeping advances in computer vision, vision systems today do not readily generalize to new tasks without extensive supervised training, and they do not have access to the detailed compositional interpretation of visual stimuli afforded to us by our own vision systems. Recent work further indicates that failure to achieve detailed interpretation is a fundamental limitation of feedforward systems (Ben-Yosef et al. [2015]).

Alignment of partial information refers to the policy where subsystems share partial results of computations throughout an ongoing computation, in contrast with staged architectures where information is passed onward only upon completion of a processing stage. In order for the sharing policy to succeed, computational elements must be able to align and integrate updates from their neighbors with the elements' own internal states. The alignment should allow elements to enhance, accelerate, or direct their own ongoing computation. The alignment policy is inspired in part by the work of Ullman [1991].

The requirement that such alignment pervade a **multimodal network** ensures that the power of the entire perceptual apparatus may be brought to bear on any stimulus, regardless of whether the stimulus itself is multimodal or if it presents in one sensory modality alone. Visual machinery can then recruit machinery used in processing other modalities in order to interpret purely-visual stimuli. Coen [2006] recognized the disambiguating power of stimuli that occur simultaneously in different modalities, and evidence of cross-modal perception abounds; the McGurk effect (McGurk and MacDonald [1976]) in which vision affects perception of speech sounds is one well-known example. The same alignment mechanism that recruits processing from other sensory modalities may also exploit contextual constraints and regularities within the visual mode, by pruning unlikely interpretations and suggesting interpretations for ambiguous visual stimuli based on contextual cues.

The alignment hypothesis can help answer the questions of how to build computational models of our visual competence, and how to build AI vision systems that share our deep analytic capacity for visual phenomena. In particular, the alignment hypothesis makes strong predictions about the types of computational models that would most successfully solve visual problems. Such models would consist of many components that integrate information from multiple sources, each flexibly handling fragmentary information, performing small steps of a distributed computation and then eagerly forwarding the partial results to neighboring nodes. Such distributed computation would likely benefit from bidirectional information sharing, with nodes communicating in a common language of representational elements via their shared interfaces. Global problems would be solved by the sustained local interactions of components across such shared interfaces, so long as the process converged to a solution.

The computational properties called for by the alignment hypothesis point toward the family of relaxation algorithms. Relaxation algorithms define global computations solely in terms of local interactions. The interactions move the global configuration incrementally toward consistency under certain constraints, or toward optimality while respecting constraints. Due to the strong locality present in images and video, relaxation algorithms have found many successful applications dating back to early work on estimating shape from shading (Horn and Brooks [1989]), and on discovering global interpretation of images from constrained local interactions

(Waltz [1972], Hinton [1978]). Relaxation algorithms intrinsically fit asynchronous, parallel, and distributed models of computation as a result of their distributed state and localized communication. The natural fit to parallel architectures has the welcome side effect of increasing scalability and biological plausibility of many relaxation algorithms. The desirable properties of relaxation algorithms thus guide the search for the best mechanisms with which to test the alignment hypothesis.

This brings the analysis to the point of concrete implementation. My goal has been to gather evidence in favor of the alignment hypothesis, by showing that systems built upon the computational foundations circumscribed by the alignment hypothesis exhibit versatility, flexibility, and introspection characteristic of natural vision systems. What mechanisms possess the desired computational properties? In Section 1.2 I present several candidates, from which I identify two mutually compatible alternatives that I develop further: propagators and neural networks.

In this section I have motivated my thesis in a way that is inspired by Marr's three levels of process understanding (Marr [2010]). I identified three aspects of natural vision, that the state of the art in computer vision falls short of modeling: the aspects of *versatility*, *flexibility*, and *introspection*. I made four observations pertaining to vision: its fundamental problem of ambiguity, its interaction with concurrent modalities, the constraints and regularities of its environment, and its goal-directed origins. I described how the observations elucidate vision's computational imperative and I presented my hypothesis that robust perception requires pervasive alignment of partial information throughout a multimodal network. This **alignment hypothesis**, and the evidence I have gathered in support of it, represent a step toward a computational theory of vision that accounts for natural vision's versatility, flexibility, and introspective capability. The alignment hypothesis also points toward a family of algorithms, relaxation algorithms, as fertile ground for discovery of specific algorithms for robust vision, and their empowering representations. In the next section I discuss candidate implementation mechanisms that fit into my motivational framework, outlined in Figure 2.

1.2 Mechanisms of alignment

The alignment hypothesis introduced in Section 1.1 has implications for choices of algorithms and mechanisms in models of vision. Because the best way to evaluate the alignment hypothesis is to build a working system whose components and internal interactions can be analyzed, I adopt the methodology of using the alignment hypothesis and its algorithmic and mechanistic implications as guides for building systems, with the aim that observed robust performance of the systems yields evidence in favor of the alignment hypothesis. Such evidence would suggest that the alignment hypothesis serves as a promising start in the greater endeavor to find a computational theory of robust vision. As a first step, I evaluate several mechanisms as potential testing grounds for the alignment hypothesis: probabilistic graphical models, restricted Boltzmann machines (RBMs), propagator networks, and neural networks.

1.2.1 Propagator networks

Propagator networks include a variety of mechansims stemming from early work by Sussman and Steele [1980] that share a common style of information plumbing. I focus on propagator networks of Sussman and Radul [2009]. These propagator networks consist of independent stateless

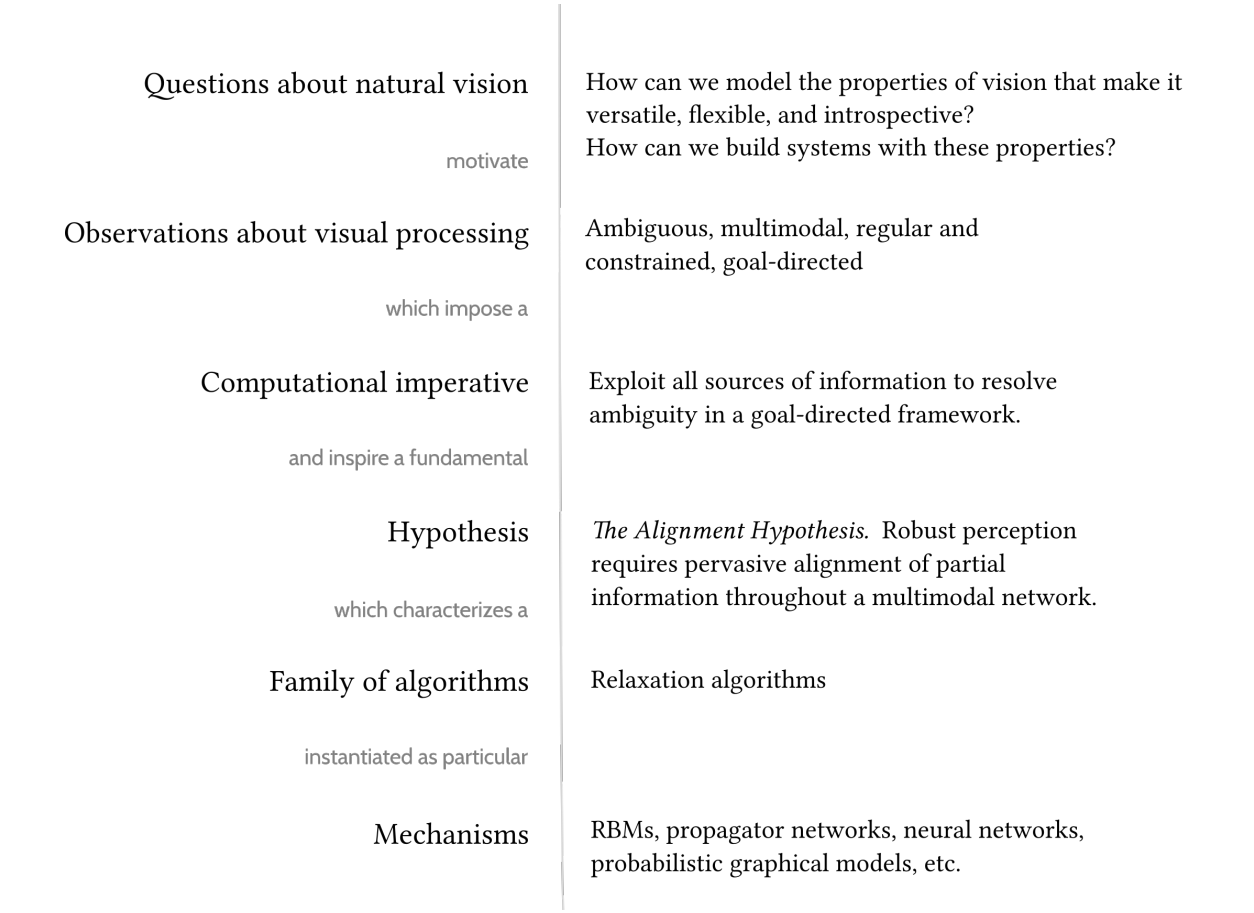


Figure 2: Motivational framework for the work described in this document

machines connected by means of cells that store state. A distinguishing property of propagator networks that sets them apart from most programming systems is the way that cells' states are updated: whereas variables in programming systems are typically updated from a single source at a time, cells in a propagator network can receive multiple updates asynchronously, in such a way that the information stored by the cell monotonically increases. As a concrete simple example, consider a scalar, real-valued cell c. An update u_1 imposes the constraint $c \geq 3$, which increases the cell's specificity by removing all values that violate the constraint. Another update u_2 imposes the constraint $c \leq 3$, resulting in the cell's value becoming maximally specified as the set containing just 3. In this example, it does not matter which order the updates are applied. The cell's connected propagators are notified whenever its value is updated, which can cause the increase of specificity in c to propagate elsewhere in the network, potentially resulting in further specificity elsewhere in the network and in c itself. By storing each update along with its computational provenance, the network can additionally answer questions about *why* a cell does not contain a particular subset of its maximum-allowable domain: the question *why does* c *not contain* 4*?* produces the answer *update* u_2.

The propagator architecture has several clear advantages as an alignment mechanism and as an engineering framework. Autonomy of the individual cells and propagator units permits straightforward parallelism. The monotonicity requirement of cells guarantees that propagators

can only increase the information content of their connected cells. Here, information content is used informally to refer to the degree of specificity of cells' values. This monotonicity requirement, combined with the declarative nature of propagators that operate by transforming and applying constraints, neatly factors apart questions about *what* components in the network do from *how* computation is scheduled in the network. The natural bidirectionality that arises from framing computation as propagation is extremely powerful: rather than explicit inputs and outputs, propagators enforce relations among their connected cells, allowing information to flow through the propagator units in potentially any direction. The ability of cells to accept updates from several, possibly redundant, computational sources permits fast approximate computational methods to coexist with thorough, expensive methods with no loss of generality. The same multiple-source ability permits collaboration between different ways of solving a problem, for example collaboration between sensory modes.

Another potential benefit of the propagator architecture is that it provides transparency and interpretability of low-level semantics. In connectionist architectures, practitioners are limited to reasoning only in terms of statistics and in terms of geometric metaphors for high-dimensional latent spaces. The low-level implementation of such systems consists of ad-hoc plumbing filled with ostensible "semantic juice" (Donahue et al. [2016]), too murky to permit analysis of anything beyond its emergent behaviors. Contrast this semantic-juice limitation with the ability afforded to system designers by the propagator architecture, to specify fine-grained interactions between components whose semantic interfaces are derived via principled methods. In the propagator design framework, complex and sometimes unanticipated behaviors arise from the net effect of many carefully-designed simple interactions, whereas in frameworks such as deep feature learning, the net effect of the system is controlled in a top-down manner via some learning strategy, but the fine-grained interactions of components exhibit a complexity that lies outside the scope of the top-down analytic methods. Superficially, this appears to be a trade-off between coarse- and fine-grained interpretability. With carefully designed primitives, however, the high level semantics of propagator networks are interpretable and correct even when unanticipated by the design—this is a strong point in favor of propagator architectures. Furthermore, as I present in Chapter 5, it is possible and beneficial to enforce intermediate-level interpretable semantics in neural network architectures.

Emergent behaviors from complex propagator systems with local correctness guarantees point to a way that the systems can achieve *flexibility*, that stands out as unique among the mechanisms considered. Eager propagation of locally-defined relations can lead to a propagator network finding unanticipated solutions to computational problems, instantiated as patterns of information flow in the propagator network. The unanticipated solutions are subject to correctness guarantees imposed by the well-behaved local interaction of components. The system can therefore exhibit flexibility by finding new ways to apply its components to solve a problem not specifically anticipated in the design. Propagator systems opportunistically fill gaps in local descriptions whenever the information needed to fill the gaps exists somewhere in the network, and at least one propagation path exists between the gap and the source information needed to fill it. Flexibility arises when the normative behavior of a system is interrupted by such a gap, but the gap is filled by opportunistic propagation.

As a concrete example of flexibility arising from opportunistic gap filling, suppose a system is designed to use stereopsis to infer depth, and then use the resulting depth measurements combined with the image to detect objects and estimate their 3D location. A gap in this compu-

tational path is formed when one image sensor is occluded, so that stereopsis fails. If an object with a narrow range of possible sizes, such as a human, is present and strongly detected based on appearance alone, then a suitable propagation-enabled object-detector could estimate the depth measurements in the vicinity of the object. The 3D-location propagator can then make progress on estimating the human's position, even though its usual source of depth information is unavailable. Implementing such flexibility constitutes an essential step toward modeling natural vision's ability to adapt quickly and with few examples to novel tasks.

One challenge in applying propagator architectures to vision is in preserving their many desirable attributes while scaling them to typical vision problems, with easily billions of sensor measurements and pervasive uncertainty. My work on this problem is the subject of Chapter 4.

1.2.2 Probabilistic graphical models

Probabilistic graphical models are another class of mechanisms that has seen widespread use in vision problems. Probabilistic graphical models are not a specific mechanism, but rather a class of abstractions that augment concrete computational mechanisms. Abstractions in the class provide structured ways to reason about the uncertainty of variables within the mechanisms that they augment. Such abstractions, which include Bayes Networks, Markov random fields (MRFs), and many others, use vertices in graphs to represent random variables, and edges to encode structure of conditional dependence, so that the resulting graphs represent families of probability distributions defined by common factorization structure. The abstractions and their associated exact and approximate inference and learning algorithms have found widespread application in computer vision. Probabilistic graphical models have appeal from the perspective of the alignment hypothesis because of the way the underlying graph-structure captures locality through distribution of state. Distributed state makes these models amenable to approximate inference via relaxation algorithms.

Graphical models have been successfully applied in multitask and multimodal alignment problems. Many examples of prior art exist but I call attention to work which demonstrated that a generalization of the Viterbi algorithm (Viterbi [1971]) applied to a unified hidden Markov model (HMM) representation for object detection, tracking, and event recognition can exceed the performance of the components in isolation (Barbu et al. [2012]). Further work showed a very similar approach applied to multimodal search problems in vision and natural language (Barrett et al. [2016]). Such successes in multitask and multimodal perception using graphical models benefit from pioneering work in grounding language in perception (Siskind [1995]) and event recognition (Siskind and Morris [1996]).

By design, probabilistic graphical models are skin-deep representations: their homogeneity enables the same suite of algorithms to apply to many problem domains, but limits representational power over any particular problem. Typically, representational power is limited to statistically-measured conditional dependence among the variables that make up the inner workings of instantiated representations. Statistical methods tend to require a lot of data to describe, even superficially, the detailed causal relationships which are nevertheless easy to describe in the right representations. Seen from this perspective, probabilistic graphical models unsurprisingly perform best when grounded in carefully designed representations resulting from keen insights, such as those by Siskind et al., into the problem's domain[1]. For this reason, in my work I focus

[1]This appraisal of graphical models' limitations applies to the classical machine learning framework. Feature

more on representation design and less on graphical models, guided by the principle that graphical models can augment good representations, but applying them too early can compensate for and obscure the flaws in bad representations.

1.2.3 Restricted Boltzmann machines

An RBM is a type of stochastic, recurrent neural network (Hinton et al. [2006]). In an RBM, a set of visible units and a set of hidden units form a bipartite graph with weighted edges. As in conventional neural networks, a node i accumulates weighted activation $a_i = \sum_j w_{ij}x_j$ from its neighbors j. The nonlinear function in an RBM node must produce a valid probability. This probability is defined as $p_i \equiv \sigma(a_i + b_i)$, where $\sigma(x) \equiv 1/(1 + exp(-x))$ is the sigmoid function and b_i is the bias for node i. Each node i outputs 1 with probability p_i, or 0 with probability $1 - p_i$. Unlike conventional neural networks, RBMs have undirected edges. When an input (assumed for simplicity of explanation to be a binary vector) is presented to the network on the visible nodes, the hidden nodes are updated according to the update rule. The values of the hidden nodes can likewise be used to update the visible nodes. Iterating hidden and visible updates forms the basis of the contrastive divergence training algorithm (Hinton [2002]), a method of updating weights using very few rounds of Gibbs sampling. The contrastive divergence method is efficient and works in practice despite having few theoretical guarantees (Hinton [2010]).

RBMs have several properties that make them good candidates for alignment mechanisms. The distributed state, and relaxation learning and inference algorithms of RBMs fit the profile of mechanism attributes that I identified in Section 1.1. Furthermore, RBMs—and closely related deep RBMs and deep belief networks—have shown great promise in multimodal learning. Hinton et al. predicted that certain multimodal architectures based on RBMs or deep belief networks could result in improved high-level features (Hinton et al. [2006]). In particular, the authors described an architecture in which sensory modalities are processed by deep RBMs or deep belief networks that share an undirected associative memory at the top layers. In practice, instances of such architectures effectively learn generative models of word associations with images that can be used for classification and retrieval (Srivastava and Salakhutdinov [2012]) and multimodal classification on audio and video where an unsupervised model trained on both modes and fine-tuned on one mode is able to classify using the alternate mode (Ngiam et al. [2011]).

The opacity of RBMs and conventional neural networks makes designing and debugging systems with these representations difficult. Whereas conventional neural networks have matured as a technology, spurred on by powerful and freely available engineering tools such as TensorFlow (Abadi et al. [2015]), development with RBMs remains relatively obscure at the time of this writing. Due to their scalability potential, RBMs may eventually come back into fashion, causing a similar excitement leading to proliferation of tools to that which has accompanied the growth of deep learning with conventional neural networks. In that future, RBMs would be worth revisiting as potential alignment mechanisms.

1.2.4 Neural networks

Conventional neural networks, particularly feed-forward deep convolutional neural networks (DCNNs), appear at first glance to have fewer desirable attributes for testing the alignment hy-

learning frameworks are a more complicated subject, explored in Chapter 2.

pothesis than RBMs, propagator networks, or probabilistic graphical models have. During inference with DCNNs, information flows unidirectionally through a staged directed acyclic graph (DAG) rather than bidirectionally and asynchronously. Stochastic gradient descent combined with backpropagation, which is by far the most successful and widely used learning strategy, generally requires global sequencing of operations and is therefore not conducive to solution via relaxation algorithms. Despite these apparent incompatibilities, neural networks are flexible and adaptable enough to implement alignment architectures. Surprisingly, such adaptations of neural networks lead to performance on par with the state of the art on tasks such as depth reconstruction from RGB images, and to networks that are able to solve new problems as a result of having internal signals with meaningful and interpretable semantics. My work on extending neural network architectures is the subject of Chapter 5.

1.3 Testing ground

> "You can't think about thinking without thinking about thinking about something."
> – Seymour Papert

Discovering good computational models of a problem-solving ability requires first finding good problems to solve. Experience has shown that toy problems, where low-level perceptual tasks are oversimplified, simulated, or hand-annotated, do not generalize to real-world visual tasks. The greatest successes of computer vision to date have been on tasks with real-world perceptual inputs, but with a limited range of outputs, such as object labels and bounding boxes, event labels with spatial and temporal extents, or semantic segmentation with a limited number of categories. Although solutions to such problems have clear economic value, the way the problems are framed prevents the solutions from addressing the questions that motivate my work: how can we model natural vision's versatility, flexibility, and introspective capabilities, and how can we build systems with those abilities? To take steps toward answering these questions, I consider several possible testing-ground problems and point out a hazard associated with choosing the wrong testing-ground problems and performance metrics.

1.3.1 Problems not to use, and why

The problem of image captioning, defined as emitting a textual description of an image's content, appears at first to be a difficult problem for which a solution would necessarily demonstrate nuanced visual understanding. Sensational results in this problem area have been covered with some astonishment by mainstream media (Markoff [2014], Dent [2016]) because the depth of thought processes that humans must engage in to produce such descriptions tempts us to believe that the systems likewise deeply understand the images that they generate captions for. The occasional but nonsensical failures of captioning systems reveal how the captioning problem does not adequately constrain systems to develop robust visual intelligence. Nonsensical failures on clearly human-interpretable images rule out that captioning systems possess human-level image interpretive capability, indicating that the success of these systems on the majority of their inputs stems from the overwhelming statistical regularity of human-generated captions appearing in the training data. Mapping images to plausible captions based on training with many pairs of images and human-generated captions is thus significantly easier than understanding images, because

most of the image understanding process is contained in the annotators of the training data. Such illusory visual intelligence, in which the illusion is revealed only by striking counterexamples, makes image captioning problematic as a testing ground for models of robust vision.

One way to address problems with captioning is to impose constraints on the types of outputs that a system may produce by having the system answer specific questions. Visual question answering (VQA) (Antol et al. [2015]) provides such a way to direct a system's attention toward a specific task in order to evaluate the system's visual intelligence. Antol et al. define VQA in terms of a benchmark so that performance can be measured programmatically. The benchmark definition of VQA is convenient for quantitative evaluation but invites the temptation to model visual intelligence via processes that exploit statistical bias in the dataset of question-image-answer triples in order to map question-image pairs onto plausible answers. Learning VQA by exploiting such bias risks incurring the same hazard that plagues image captioning systems: frequent good performance on problems that appear cognitively challenging can create a false sense that systems model our cognitive abilities, whereas even a single unambiguously wrong output suffices to demonstrate that they do not. Furthermore, it is difficult to obtain actionable information about how to debug and improve systems from the occasional nonsensical outputs of statistically learned models.

Illusory cognitive modeling resulting from over-reliance on statistical learning creates a hazard in problems like VQA and image captioning, and draws emphasis to a subtle but important consideration in selecting the problems and evaluation methods for development of models of visual intelligence. Successful deep-learning models with very large capacity show excellent measured generalization performance on test sets. Such models are demonstrably capable of severely overfitting certain pathological datasets (Zhang et al. [2016a]) with an overwhelming efficiency that suggests that some amount of memorization of training data is a reality of all deep learning systems. The findings of Zhang et al. suggest that strong precautions must be taken to prevent such memorization from having negative consequences, that remain completely hidden by current best practices in measuring generalization performance.

Precautions against overfitting cannot protect against the consequences of buggy data in which a bias affects the test data and training data alike, nor can the precautions protect against implicit bias arising from the specifics of the training problem. As I present in Chapter 2, implicit bias towards brittle or incomplete cognitive models, baked into the problems we ask deep learning systems to solve, can manifest even in straightforward image classification tasks. In many but not all commercial applications of machine learning, the training tasks are identical to the task of commercial interest, the training data are vast and cover all plausible operating scenarios, and occasional failures do not put life or liberty at risk. When the training data sparsely cover the operating domain, or when the training task oversimplifies or embeds bias about the real-world problem, attempting to mirror the input-output characteristics of high-level cognitive abilities via statistical training becomes inaccurate. Making matters worse, such inaccurate modeling becomes morally reprehensible in zero-tolerance applications in which creating the illusion of intelligence could convince a reasonable person to trust the system as though its abilities were genuine. Although the moral argument is outside the scope of my thesis, it has never been outside the purview of AI engineers' responsibilities.

Practical considerations about at what level statistical modeling is appropriate bear heavily on my work, as do choices about problem framing that support development of models of human intelligence rather than convincing but fundamentally-flawed mimicry of the products of human

intelligence. Even in cases where such mimicry succeeds due to overwhelming availability of training data, it leads to intellectually unsatisfying models. Had physicists adopted the dismal worldview that nature had no more to offer, by way of insight, than statistics dutifully recorded in vast quantity from experiments, they may still have produced large and impenetrable models capable of predicting the motion of objects in the world as well as the elegant laws of classical mechanics do. They might have said, *"Nature is obviously a vast complicated system. We have no reason to expect simple explanations for its workings."* The only way to reveal such a statement as a failure of imagination is to find just one better model, for example, $\boldsymbol{F} = m\boldsymbol{a}$. Scientific endeavor to understand human intelligence likewise thrives on perseverance to find simple and elegant models, and optimism that such models exist to be found.

1.3.2 Problems to use, and why

To avoid conflating models of statistical regularity in input data with models of the cognitive processes that produce such data, I re-framed VQA as a Turing Test analogue for visual intelligence, in which a human engages the system in an interactive question and answer session to expose gaps in a system's visual intelligence. Framing VQA as an interactive test makes quantitative performance metrics difficult to develop, but this difficulty is of little detriment to the endeavor of discovering models of vision's computational processes, in which a premature focus on optimizing a numerical measure of performance would curtail a thorough exploration of possibilities. I did not model the language aspect of VQA; rather, I sought representations that can empower natural language systems such as Genesis (Winston [2014]) to answer questions about visual scenes. I focused on modeling the introspective capability of natural vision by designing representations that provide question answering systems with the ability to provide support and explanations for their answers. I developed and qualitatively evaluated such representations in the context of grounding scene geometry in observation of human movements, using a framework inspired by Sussman's propagator architecture to establish bounds on estimates and to track support of knowledge in the system. Given a video recorded from a fixed vantage point in an outdoor urban environment, the system supplies information about objects and their 3D location and movement, as well as background occluders in the scene. This work, which is the subject of Chapter 4, is a step toward perception systems that will empower action recognition in story understanding systems.

My work on using propagators to bootstrap understanding of scene geometry made use of coarse-grained primitives such as background subtractors. This implementation was a first step toward evaluating the alignment hypothesis in a real-world scenario. The next step was to take the implementation further toward *pervasive* alignment, in which information propagates from scene-level reasoning all the way through to the low-level descriptions used internally by the components themselves. The coarse-grained components that I used in the initial implementation that I describe in Chapter 4 are not designed to have this capability, so I sought to design mechanisms that have such alignment capabilities. The development effort led me to deep neural networks, whose success in object detection and generative modeling of images continues to overwhelm that of other mechanisms. In order for neural networks to align information from external sources with their internal descriptions, the networks need to share common representations across their interfaces with other components. In Chapter 5 I describe my implementation of such networks, applied to the problem of reconstructing distance from focal plane (depth) from

single monocular RGB images. I chose the depth reconstruction problem as a testing ground for such low-level alignment because it is complex enough to be interesting while remaining feasible under the constraints of computational complexity and data availability that present major challenges in deep learning. In addition, the depth reconstruction problem has clear applications to higher-level scene understanding. A surprising result from my work on depth reconstruction using networks trained to learn representations with interpretable semantics is that these networks perform on par with state of the art depth reconstruction systems for cluttered indoor scenes. Another surprising result is that the networks have the ability to dynamically respond to external signals that simulate multi-modal influence, in order to reduce the severity of errors.

1.4 Overview

I have motivated my work in terms of what sets natural visual abilities apart from the abilities of the best performing computer vision systems today. I asked how we can capture natural vision's ability to perform robustly under widely varying conditions, its ability to adapt quickly and with little training to new tasks, and, especially, its ability to provide us with rich compositional explanations of its findings. I made four observations that pertain to the environment's influences on vision systems: vision is fundamentally uninvertible, vision is just one component of a multimodal perceptual system, the natural world is rich with constraints and regularities, and vision evolved to serve survival goals. These observations led to a hypothesis about the high-level computational organization of natural vision that enables its unique abilities. This **alignment hypothesis**, in which I anticipate that **robust perception requires pervasive alignment of partial information throughout a multimodal network**, then informed algorithmic and mechanistic choices in designing vision systems. In particular, the hypothesis draws attention to relaxation algorithms, and indicates that propagator networks and certain types of neural networks are good mechanism choices.

In order to gather evidence in favor of the alignment hypothesis, I built systems upon the computational foundations circumscribed by the alignment hypothesis, seeking to validate that the systems possess the desired robust attributes. I identified two problems related to reconstructing scene geometry from images to use as testing grounds for implementations. The first is the problem of reconstructing scene geometry, determining locations of occluders and affordances such as benches, and 3D location and orientation of surfaces in crowded urban scenes. The second is estimating depth maps from single monocular images of cluttered indoor scenes. I identified a hazard by which certain approaches, which are common in the machine-learning discipline, encourage modeling of superficial statistics of datasets that can create the illusion of visual intelligence, but that embed biases that cause brittleness. I took care to avoid this hazard in my own work.

In Chapter 2 I investigate a type of neural network performance anomaly that relates brittleness and task-embedded bias as I discussed in Section 1.3. Chapter 3 is an overview of propagation, relating several prominent historical uses of visual propagation to my own work. Chapter 4 details my efforts to build a visual propagator system that can make inferences about the geometry of a scene that it monitors continuously with a stationary camera. In Chapter 5 I present my work on a related problem, estimating depth from images, that draws on findings of the experiment described in Chapter 2 and the outcomes of the work I describe in Chapter 4. In the appendix I present some useful techniques I developed in the course of my work with neural

networks. I summarize my contributions in Chapter 6.

2 Characterizing Neural Net Classification

In this chapter you learn about several experiments I performed to investigate and characterize neural network classification. The results of the experiments point to representational shortcomings of neural networks that I alluded to in Section 1.3.1, and motivate development of alignment-based methods to overcome the shortcomings.

The experiments in this chapter use a technique I developed to remove from natural images all but the features most essential to their classification by neural networks. The technique works by reducing image signal energy without reducing neural network classification confidence. Examples of the inputs and outputs of this technique are shown in Figure 3. A surprising result presented in in this chapter is that the recognizability by humans of the reduced-energy images depends on the class of object in the image more than it depends on other image properties.

(a) Input natural images

(b) Output reduced-energy images

Figure 3: Reduced-energy images that a neural network recognizes
A neural network classified the natural images in (a) with high confidence as *meerkat, Russian wolfhound, marimba, remote control, guinea pig*. The images were processed by an algorithm that removed as much signal energy as possible while preserving the original classification confidence reported by the neural network. The results, shown in (b), exhibit varying degrees of recognizability to human observers.

2.1 Introduction

My work toward achieving robust vision models by pursuing the alignment hypothesis of Chapter 1 has focused on two mechanisms: propagator networks and neural networks. Propagator networks, the subject of Chapters 3 and 4, achieve robustness through complex emergent properties of many simple local interactions that adhere to a principled design. The rules of local interaction impart their explanatory power on emergent behaviors of the system as a whole. Neural networks, by contrast, enforce constraints in terms of loss functions evaluating a complex behavior to be learned, but do little to constrain how the behavior is implemented by the individual components of the network. Experiments have shown that this leads to a different and unprecedented type of robustness: the ability to extract, from large collections of data, hierarchical patterns that are empirically more descriptive than previous carefully-engineered representations. In Chapter 1 I alluded to a flaw in the apparent panacea of neural networks: the conspicuously nonsensical failures that plague these systems. In this chapter I present an investigation of that flaw, and its implications on future work in building robust systems with neural networks.

Despite their unprecedented success in, for example, classification tasks, deep convolutional neural nets (DCNNs) exhibit several related flawed behaviors that collectively demonstrate *brittleness*. A subcategory of such brittle phenomena is *fooling by adversarial examples*: a process by which an adversary generates examples constructed specifically to fool a neural network. The adversary can fool the DCNN into confidently misclassifying images that bear no resemblance to the reported category or to any natural image, as demonstrated by Nguyen et al. [2015], or into confidently misclassifying images after an imperceptible signal has been added to an original, correctly-classified image, as demonstrated by Szegedy et al. [2013].

Adversarial examples point to a serious flaw in current high-performing DCNN models, in that their capacity to fool DCNNs proves definitively that DCNNs do not yet exhibit robust visual intelligence. Mounting evidence suggests that adversarial fooling is a problem with widespread implications: Moosavi-Dezfooli et al. [2016] found a method of constructing perceptually-irrelevant perturbations that cause misclassification when added to nearly any image. Their method *generalizes across neural networks*. Koh and Liang [2017] extended the finding to training examples: even single adversarial training examples within a large corpus can cause the resulting trained classifier to fail on specific unmodified test images. In Section 1.3 I argue that given the bias of statistical learning systems toward sophisticated but flawed mimicry of intelligent systems' outputs, even isolated unintelligible failures rule out the possibility that such learned systems are robustly intelligent. It would be convenient to be able to dismiss adversarial fooling as a quirk that negligibly impacts real-world performance. To dismiss such profound contradictions of robust visual intelligence responsibly, it is crucial to understand adversarial fooling well enough to rule out the possibility that it is not merely an indicator of a larger class of brittle phenomena.

In this chapter, I present experiments to shed light on potential underlying causes of fooling by exposing what visual features in natural images are most essential to image classification by neural networks. I believe that the problem of interest is not *that* neural networks can be adversarially fooled. The problem is that we do not understand *why* in a way that yields insight. Extensive work has been done to characterize several types of fooling phenomena, including many sufficient conditions to cause fooling (Nguyen et al. [2015], Szegedy et al. [2013]). The putative cause of certain fooling phenomena is the compounding effects of linear units in deep nets, combined with the counter-intuitive fact that in high-dimensional spaces, vectors may have

large magnitude yet have only very small components parallel to the individual vectors in a given basis (Goodfellow et al. [2014b]). This leads to imperceptible perturbations in image space having large effects on classification performance. This low-level account of neural network fooling leads to generalizations about fooling techniques, but it does not shed light on how the emergent behavior of neural network mechanisms gives rise to the fooling phenomenon, and it does not provide a clear set of strategies to build and train networks that cannot be fooled.

Humans can be adversarially fooled too, but we know that optical illusions really do represent quirks that negligibly impact real-world performance. On the contrary, the nature of such quirks helps shed light on how vision works. In many cases, the human vision system contextualizes illusions and creates descriptions of them with the same rich, compositional detail that it uses when describing non-illusions. In the case of the illusions, the compositional explanations can even yield self-reflective insight into how we process images. The illusion of a foreground shape in Figure 4 inspires a striking example of this self-reflection process.

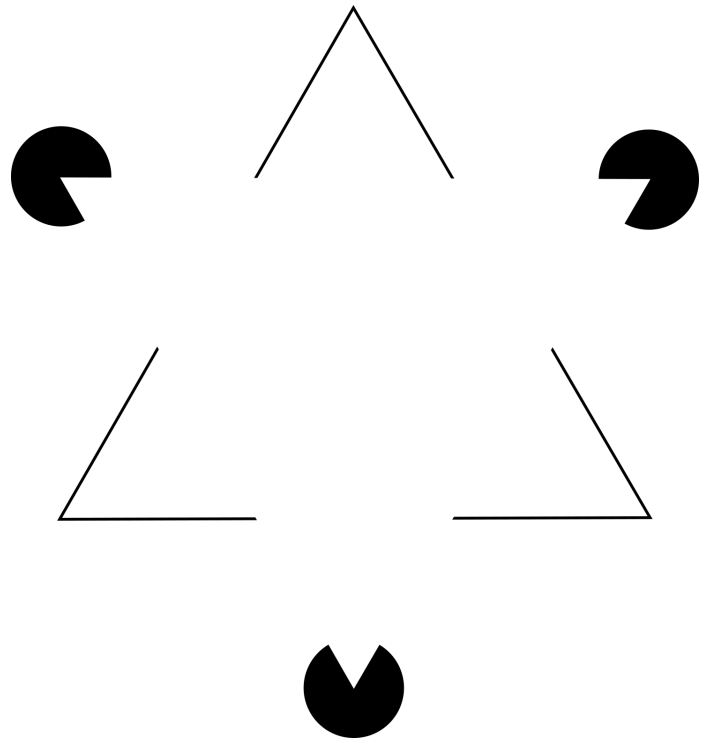


Figure 4: The Kanizsa Triangle illusion
Adversarial examples for human visual systems point out positive attributes, rather than bugs, of visual intelligence.

Rather than focus directly on fooling in my own experiments by asking, as others have asked, *what conditions and mechanisms cause confident misclassification of certain images by neural nets?*, I ask instead *what are the essential features of natural images that result in their confident classification by neural nets, and how do those features differ from the features that are essential to classification by humans?* In this way, I approach the fooling phenomena with the opposite perspective from that of prior work on the subject. I implement this approach by reducing natural images to their essential features necessary to maintain neural network classification performance, thereby

exploring a smaller—though still very large—space of potential fooling images than would be explored with a less constrained image-generation technique. Critical questions to address in my approach are *do minimal sets of features that support strong classification by neural networks also suffice to support strong classification by humans? If not, how can we account for the differences?*

I approached the problem of generating minimal feature sets that are recognized by deep neural nets with high confidence by using a greedy algorithm to adversarially remove signal energy from images. Specifically, the algorithm generated minimum-energy images through a process that starts with correctly-classified images and then minimizes signal energy in a way that does not reduce classification confidence. The results of the process are images for which any further application of my signal-energy reduction algorithm would result in a drop in classification confidence. Observations of such images, exemplified in Figure 5, shed light on the features of natural images that are most crucial to classification by DCNNs.

2.2 Methods

2.2.1 Network models and images

In order to obtain reduced-signal images that are detected with high confidence by high-performing DCNNs, I used the pre-trained AlexNet model (Krizhevsky et al. [2012]) provided by the Caffe software package (Jia et al. [2014]). This model was trained on a subset of the ImageNet dataset (Deng et al. [2009]), and though its performance lags behind the current state of the art, modern architectures suffer just as profoundly from adversarial fooling (Moosavi-Dezfooli et al. [2016]) and the small size and low latency of AlexNet made it convenient to work with. I selected images directly from ImageNet that were not contained in the training data of the 2012 ILSVRC, as specified by Russakovsky et al. [2015b], and used the images as the source material from which to generate reduced signal images via my randomized algorithm, which I describe in detail in the next section. The image selection script ensures detection confidence of 98% or higher on the source images.

2.2.2 signal-energy reduction algorithm

I used a randomized algorithm to remove much of the signal energy from natural images while preserving high classification confidence by the DCNN. The first step in the procedure is to divide the source image into its Laplacian pyramid. The algorithm then performs a search through candidate images, where each candidate is generated by randomly selecting a level from the Laplacian pyramid, excluding the image-mean level, and then randomly selecting a rectangular region within that pyramid level and setting all pixels within that region to zero. The candidates are evaluated by reconstructing an image from the modified Laplacian pyramid and using the DCNN to evaluate a label and confidence for that image. The algorithm rejects all candidate image modifications that have confidence below that of the original image or that have assigned labels different from that of the original image. The type of search used is a randomized beam search, in which a population of the most fit modified Laplacian pyramids is retained, and at each step a member of that population is randomly selected to undergo further modification. The fitness function $f(I)$ used by the search to evaluate the fitness of an image I that is not ruled out due to low confidence is proportional to negative signal energy of the mean-subtracted image,

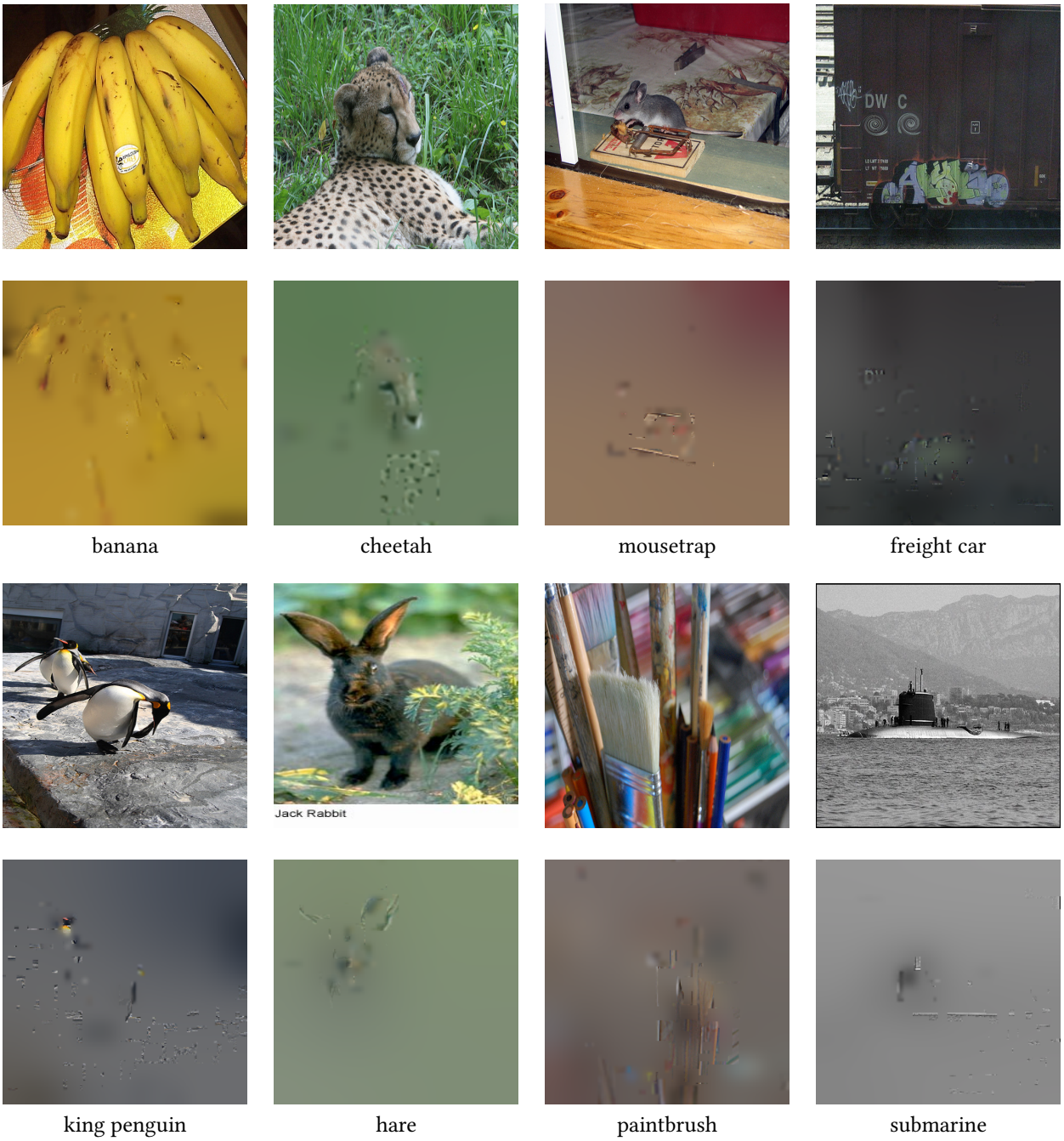


Figure 5: Example reduced-signal-energy images

Pairs of original images (top) with corresponding reduced-signal-energy images (bottom). All images are detected by the DCNN as the reported category with confidence 98% or higher and the reduced-signal-energy images are detected with the same confidence as the original images from which they were generated.

and is defined in Equation 1, where x and y are pixel coordinates and μ is the image's average pixel color:

$$f(I) = -\sum_{x,y}(I(x,y) - \mu)^2 \tag{1}$$

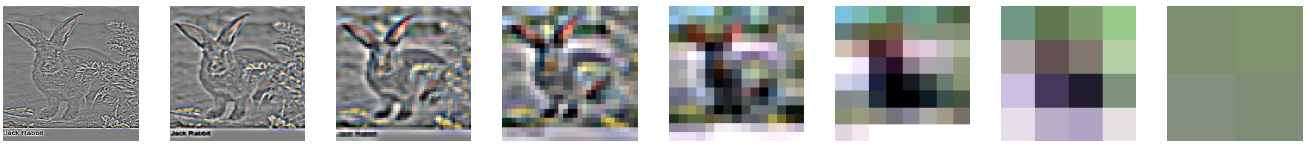
I use a Laplacian pyramid to represent the image during modification because the Laplacian pyramid conveniently facilitates removal of signal energy selectively from each of the pyramid's frequency bands. I opted to selectively remove energy from Laplacian pyramid levels in order to gain insight into the frequency distribution, in addition to the spatial distribution, of the most important features used by DCNNs to classify natural images. Figure 6 illustrates the signal-energy reduction algorithm's inputs and outputs on an example Laplacian pyramid.

2.2.3 Reduction algorithm design issues

In my work on extending the results of Nguyen et al. by generating high confidence fooling images via a large collection of different methods, I found that it is easy to devise random algorithms that can be used in combination with greedy search to generate fooling images for a DCNN. In fact, it is challenging to find image transformations that do not result in fooling images when combined with the right kind of search. Nguyen et al. showed that it is possible to create fooling images for DCNNs by directly manipulating image pixels one at a time in the HSV color space starting from a random initialization, and by manipulating compositional pattern-producing networks (CPPNs) that serve as encodings of images. I explored the fooling phenomenon further and found many more successful techniques. Some examples are depicted in Figure 7.

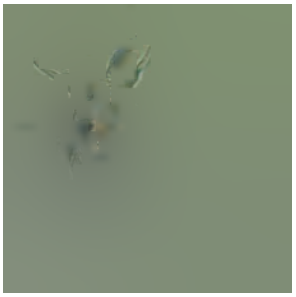
A goal in my experiments was to expose the features in natural images that DCNNs rely on most crucially for classification. To achieve that goal, I needed to prevent the effect of the signal-reduction algorithm creating new features in the image by accident. The result in Figure 7d shows that without additional constraints on the transformations used to selectively delete information in the Laplacian-pyramid levels, it is possible for those transformations to cause DCNN fooling. The results shown in Figure 7 indicate that DCNN fooling is easy to do, suggesting that care must be taken that the signal-energy reduction algorithm creates minimal images by eliminating all but essential features, rather than by creating new, incidental features. I mitigated the effect of such incidental feature creation by placing constraints on the erasure window size. When the procedure that was used to generate Figure 7d was additionally required to exhaustively search possible deletion rectangles of larger relative dimensions first before being permitted to progress to searching for smaller erasure rectangles, I found that it was no longer able to generate fooling images such as the one in Figure 7d. In practice exhaustive search is infeasible, so I required the algorithm to produce a number of unsuccessful candidate images via deletions of a certain range of window sizes before irreversibly moving to the next smallest window-size range. Combining the effect of such a monotonically-decreasing deletion window with a fitness function that does not reward increases in detection confidence beyond that of the starting image further reduces the effect of accidental DCNN fooling. The fitness function used in the signal-energy reduction algorithm does not reward changes in confidence; instead, it rewards decrease in signal energy and requires that confidence remain above a threshold set by the original image's detection confidence.

(a) Starting Image



(b) Starting Laplacian Pyramid

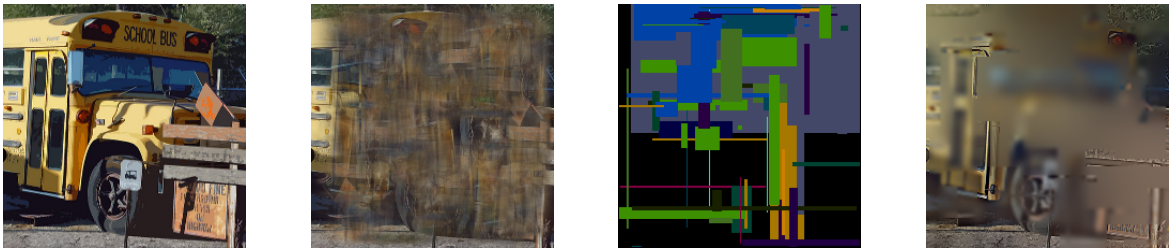
(c) Final Laplacian Pyramid



(d) Final Image

Figure 6: Signal-energy reduction steps

The signal-energy reduction algorithm starts with an image (a) and computes its Laplacian pyramid, a lossless representation of the input image comprised of 7 layers that act as band-pass filters for image signal energy, and a residual 4-pixel image containing the mean of each quadrant of the image. The frequency-selective layers of the Laplacian pyramid corresponding to the starting image are shown in (b), with visual enhancements to show contrast. The signal-energy reduction algorithm produces a final Laplacian pyramid, shown in (c). The composite image reconstructed from the final Laplacian pyramid is shown in (d).



(a) (b) (c) (d)

Figure 7: Adversarial examples generated by 3 methods
A source image (a) was used to produce three fooling images (b) - (d) that are labeled *ostrich* with 99% confidence by DCNN. Each fooling image was generated by applying a randomized beam search to maximize classification confidence on the label *ostrich*, applying one of 3 image transformations at each search step. In image (b) the image transformation was to randomly transpose a horizontally- or vertically-flipped region of (a) to another part of the image while applying partial transparency to the transposed region. In image (c) the image transformation was to start from a uniform black image and randomly select rectangles to fill with a randomly selected color from the color palette of (a). Image (d) was generated by applying the transformations used in the signal-energy reduction algorithm to (a) without any guards against accidental DCNN fooling, and maximizing confidence in the label *ostrich* rather than using the fitness function defined in Equation 1.

2.3 Results

I ran the signal-energy reduction algorithm on 142 images from ImageNet, each of which had a detection confidence via the reference AlexNet DCNN of 98% or greater. Plotting the ratios of final to initial signal energy in each Laplacian pyramid level shows that, on average, less than 10% of the signal energy contained in each of the top 5 Laplacian pyramid levels contributes to the confident detection of the image label. I note that the reduction algorithm has a bias to preserve more energy in the higher pyramid levels, because these levels have lower resolution and so it is impossible to delete rectangular regions smaller than a certain fraction of the linear dimension of these pyramid levels, because such regions would be less than one pixel wide along one or more dimensions. Ignoring the upward trend in signal energy ratio starting at level 3 of the pyramids which may be attributed to algorithmic bias, it appears the DCNN may be slightly more sensitive to signal energy in the higher frequency ranges, but the effect is not pronounced (Figure 8).

The signal-energy reduction algorithm described in Section 2.2.2 provides no guarantee that the result is unique. Many local minima are likely to exist in the landscape defined by Equation (1) under the constraint that detection confidence remains above a given threshold. It is therefore surprising that the algorithm consistently produces visually similar results on multiple independent runs on the same input image, with different random initializations. Figure 9 illustrates the similarity that is typical among results of independent runs of the algorithm on one image.

By applying the reduction algorithm iteratively it is possible to identify a nearly complete set of features that the DCNN relies on for detection. The algorithm is run on the source image to produce a Laplacian pyramid representation of the reduced-energy image. That Laplacian pyramid is subtracted from the pyramid representing the input image, resulting in a residual image that is missing all of the high-confidence features found by one run of the signal-energy reduction algorithm. The process is repeated on the residual images, using the new detection

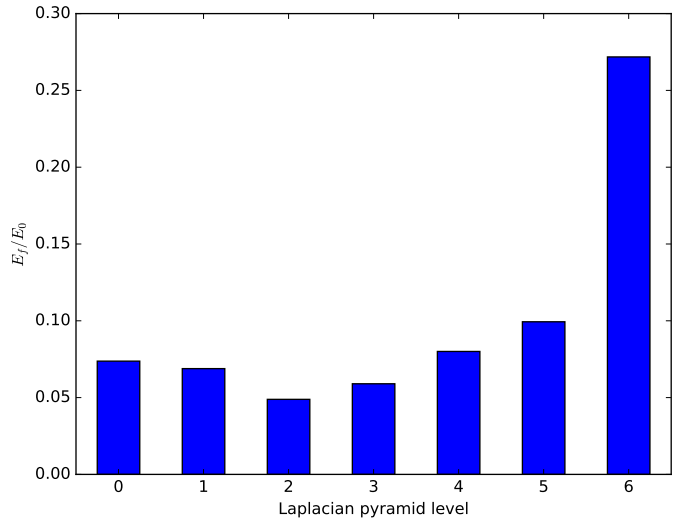


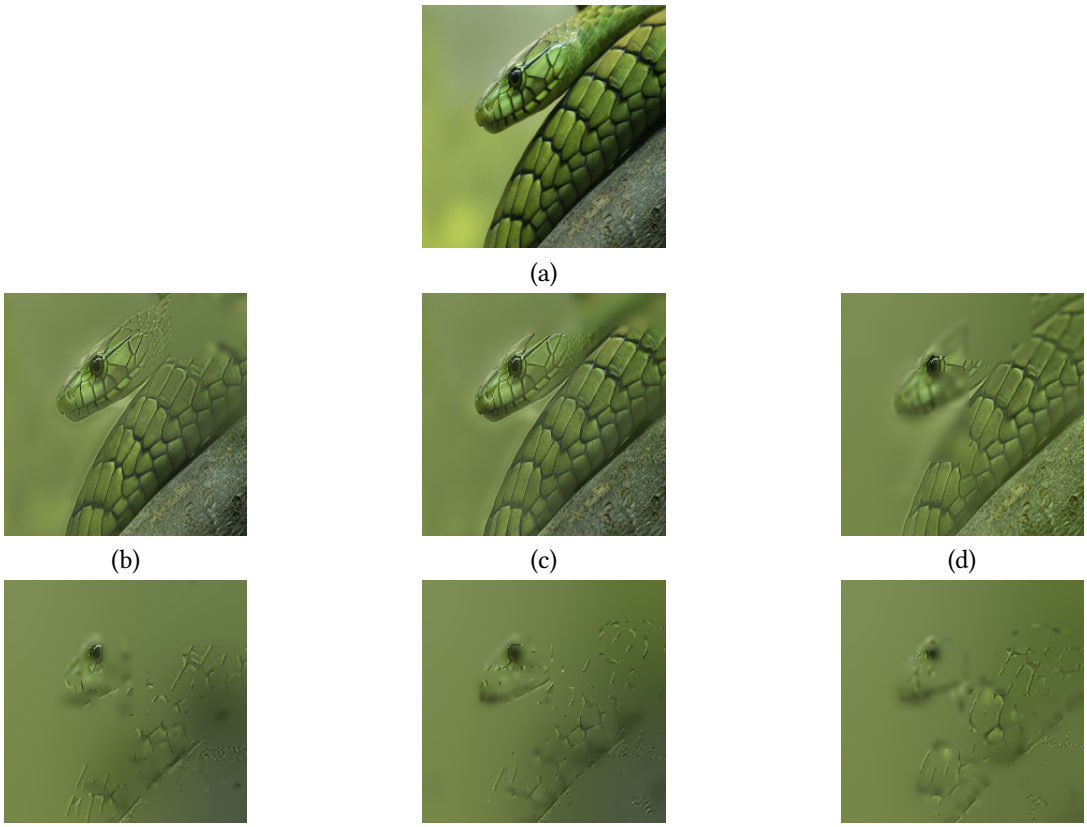
Figure 8: Energy ratios per Laplacian-pyramid level

Mean final/initial energy ratios per Laplacian pyramid level, based on 142 sample images from the ILSVRC 2012 validation data that were detected by the reference DCNN with 98% confidence or higher.

confidences, until a marked drop in confidence occurs. Figure 10 illustrates this process on an example image, and Figure 11 illustrates the composite image created by accumulating all of the features isolated by the signal-energy reduction algorithm in each iteration. By construction, the features isolated in this way are non-overlapping in their Laplacian pyramids, so the last image in the series in Figure 11 is the complement of the last image in the upper series in Figure 10 and adding their Laplacian pyramids produces the Laplacian pyramid of the original image.

An interesting observation about the reduced-energy images is that some are immediately recognizable as modified versions of the image class of the corresponding original images, while others are not recognizable at all. Of even more interest, the recognizability of images seems to generalize over images classes: the familiarity to a human observer of the results of signal-energy reduction seems to be closely associated with the label of the starting image. This phenomenon is illustrated in Figure 12. I conjecture that the DCNN uses qualitatively different strategies to detect object types that tend to be easily recognizable by humans from their reduced-energy images, versus those that are not easily recognizable by humans from reduced-energy images. The strategy used by the DCNN to detect objects that are easily recognized by humans from their reduced-energy images is to detect the objects based on the presence of features according to an appearance model that generalizes roughly to a human observer's appearance model of the object. The strategy used by the DCNN on objects that are unrecognizable by humans from their reduced-energy images is a discriminative strategy, that first rules out certain high-level features of the object and then applies a specialized mode of detection, that is sensitive to features that are only useful for detection once other categories have been ruled out.

There are many ways that a strategy based on ruling out high-level features can be realized in a DCNN. The DCNN could, for example, implement that strategy at the highest levels of the network, by creating a structure that works analogously to lateral inhibition, effectively rejecting



(a)

(b) (c) (d)

(e) (f) (g)

Figure 9: Invariance of signal-energy reduction algorithm to RNG initialization
Three runs of my algorithm with different RNG initializations on the starting image of a green mamba in (a) generated the three intermediate images in (b) - (d) and the corresponding final images in (e) - (g). The intermediate images exhibit more visual variation due to the different RNG initializations than the final images.

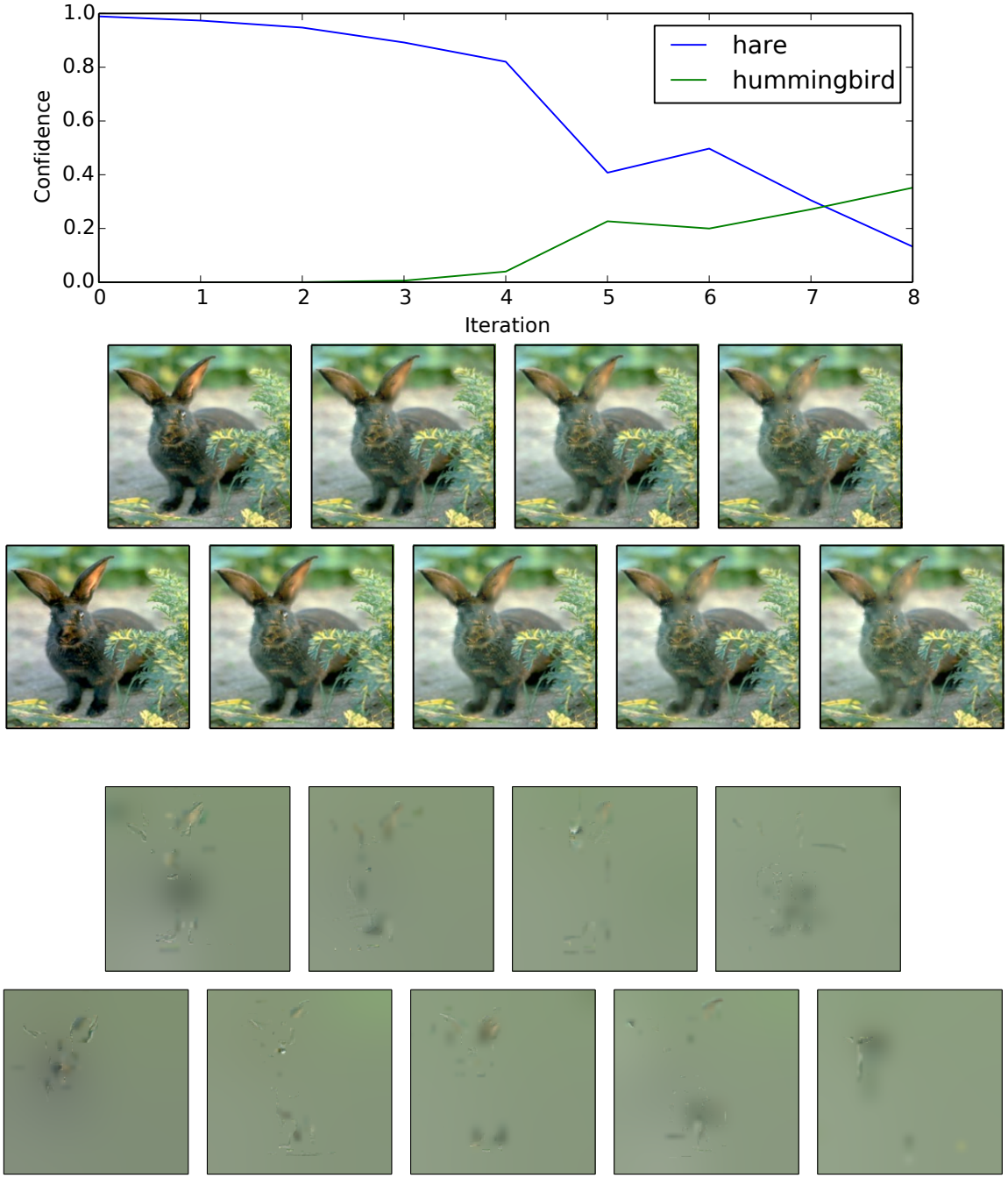


Figure 10: Residual and minimal-energy images from iterative energy reduction
DCNN confidence in two labels when signal-energy reduction is applied iteratively, where the input image in the K^{th} iteration is the residual image constructed by subtracting the Laplacian pyramid of the minimal image of the $K-1^{st}$ iteration from the input image of the $K-1^{st}$ iteration. The lower series of images shows the corresponding minimal images produced by the signal-energy reduction algorithm, i.e., a minimal set of features that the DCNN requires for classification as a hare. The upper series of images shows the residual images at each iteration, i.e., the parts whose presence was unneeded for the DCNN to be confident the image was of a hare. By the 8^{th} iteration the DCNN detects a hummingbird with 35% confidence and a hare with 13% confidence.

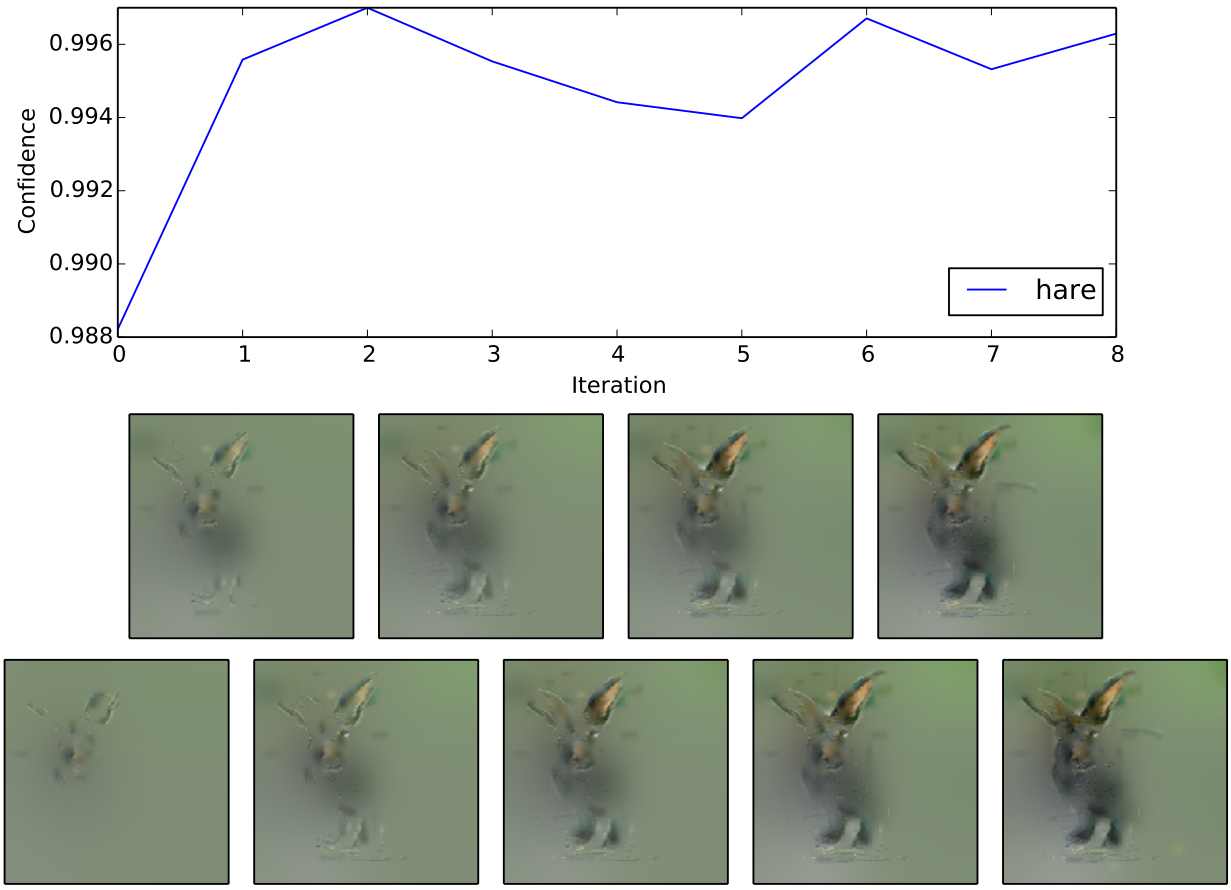
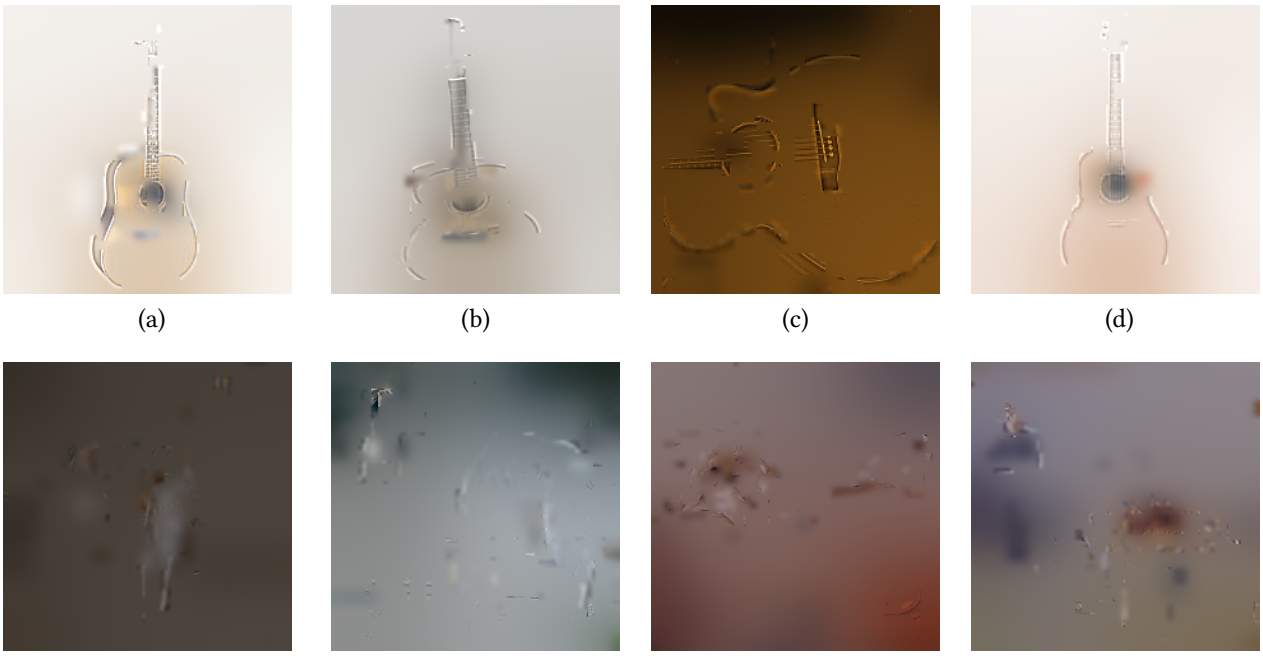


Figure 11: Accumulated minimum-energy images

DCNN confidence for a series of images corresponding to partial sums of the Laplacian pyramids of a series of serially-extracted reduced-energy images, which are shown in Figure 10.



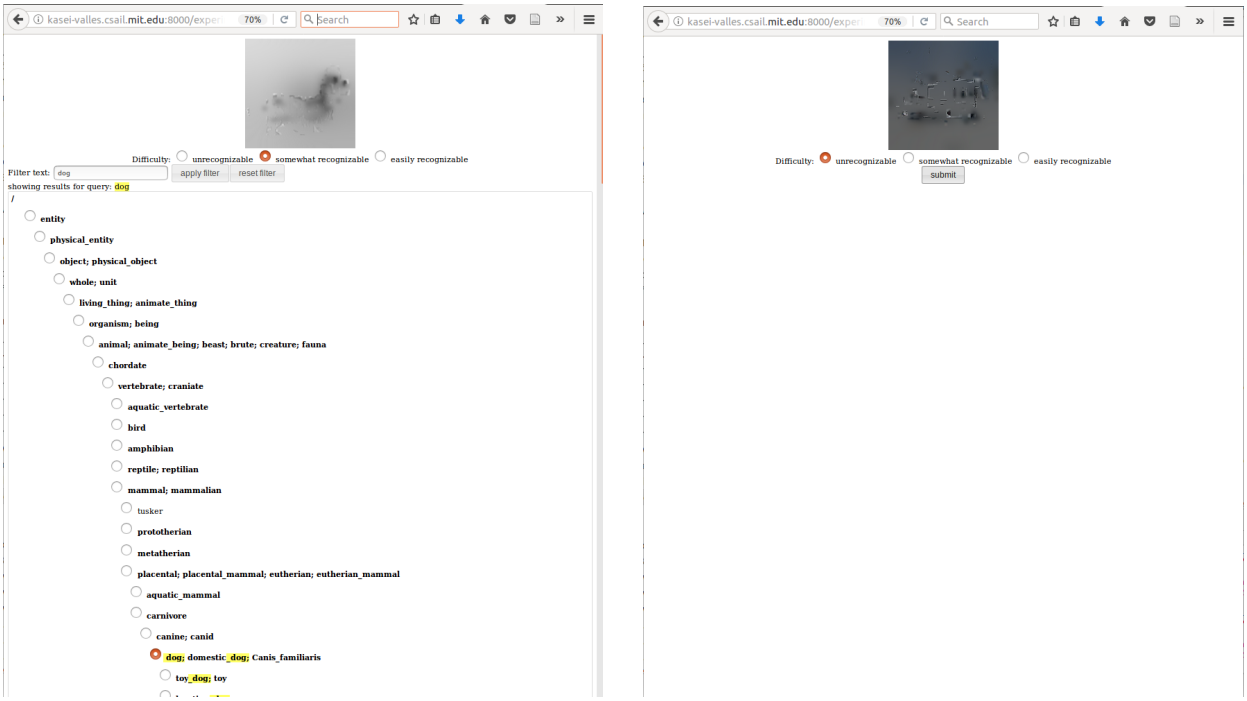
(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 12: Class-specificity of minimum-energy image recognizability
Images 12a - 12d and 12e - 12h are representative samples of minimum-energy images of accoustic guitars and Russian wolfhounds, respectively. Each member of both sets of images is classified by the DCNN with 98% confidence or above, but the minimum-energy images of accoustic guitars are easily recognizable by humans whereas the others are hardly recognizable as animals.

a certain explanation for the image if a more reliable explanation is available. One of many ways to represent such inhibition in a DCNN is before the final softmax layer, where each unit's output is proportional to the logarithm of the corresponding label's probability. A unit in this layer with a relatively high dynamic range can overwhelm units with relatively small dynamic range when outputting a positive signal, permitting a discriminative strategy in which a specialized set of features can be applied to detect a class, only when other classes have been ruled out.

I looked for evidence that a discriminative strategy realized by differences in dynamic range in the pre-softmax units could partially account for the qualitative difference in recognizability between image classes. To reduce the effect of my own biases when looking at the reduced-signal images, I conducted a small pilot study with 5 participants, who were asked to identify objects from the minimal images generated by my algorithm and assign a recognizability score to each image. Participants were shown a sequence of web pages with reduced-energy images alongside expandable trees that represented the position in the WordNet (Fellbaum [1998]) hierarchy of all 1000 ILSVRC categories. I asked the participants to guess the most specific descriptor that they were confident described each image that they saw. Additionally, I asked participants to score the image as unrecognizable, somewhat recognizable, or easily recognizable. Choosing unrecognizable caused the category-selection controls to disappear. Because expanding the WordNet tree to find a selected category would be unreasonably tedious for the subjects, the interface also provided a substring-search function. Two screenshots of the experiment interface are depicted in Figure 13.



(a) Recognizable controls

(b) Unrecognizable controls

Figure 13: Experiment user interface

The experiment's user interface allowed subjects to search the WordNet hierarchy for phrases like dog (a) and to reject an image as unrecognizable without making a choice (b). The tree shown in (a) is expandable and collapsible by clicking on its boldface nodes.

To score the results of the pilot study, I considered only two categories: recognized and unrecognized. Images were placed in the recognized category if an experimental subject rated it either somewhat recognizable or easily recognizable, and their identification of the object was above the target label in the WordNet hierarchy, and within an edit distance of 4 from the target label. For example, if the correct answer for an image was *Russian wolfhound* and the subject answered *dog* and *somewhat recognizable*, then the image was considered recognized. I chose the distance of 4 so that an answer such as *bird*, *dog*, or *snake* would correctly match a breed or species, as I could not expect subjects to know the fine-grained differences between dog breeds or many animal species. If the subject answered *unrecognizable* or if the subject's answer was not hierarchically above, or not within an edit distance of 4 from the true label, then the image was considered unrecognized. This methodology is generally a coarse, optimistic measure of recognizability. For example, a subject's choice of *instrumentality* would correctly match the target label *bobsled* under these rules. Therefore, participants were additionally asked to choose the most specific category they believed to include the depicted object, and to refrain from choosing certain general WordNet categories I judged to have little distinguishing semantic specificity with regard to image classification, such as *entity*, *artefact*, *instrumentality*, or *device*.

From the results of the pilot study I identified 7 object types that were recognized from their minimal images by all subjects, and 7 object types that were not recognized by any subjects. I collected 10 examples images of each object type that were detected with 98% confidence or above

Figure 14: Most-often and least-often recognized images by study participants
The most recognizable (top row) and least recognizable (bottom row) reduced-energy images as perceived by the 5 subjects of a pilot study.

by the DCNN. Within this sample of 140 images, I measured the maximum and minimum pre-softmax activation of the units corresponding to the 14 labels of interest. The resulting dynamic ranges are shown in Figure 15. The overall difference in dynamic range between the recognized and unrecognized categories provides evidence that the neural network uses a different classification strategy for each category.

2.4 Discussion

In this section I discuss what the results say about the descriptiveness of DCNN models. I interpret the class specificity of model descriptiveness in terms of a supposed high-level classification strategy in which the DCNN learns the 1000-way classification game, but in a way that does not support robust visual intelligence. I discuss implications that the results have for training and applying neural nets, and the implications with respect to the alignment hypothesis of Chapter 1.

2.4.1 Descriptiveness of DCNN models

Nguyen et al. [2015] hypothesized that the counter-intuitive properties they observed, such as the ease with which high-confidence false positives can be generated, the variety observed in such generated images, and the overwhelming unrecognizability of such images to human observers, may be due to the fact that the DCNN models they studied were discriminative rather than generative models. Discriminative models partition the space of images and evaluate confidence based on distance between a test image and the nearest classification boundaries. Generative models rely on density models of the training data, so that a test image would have to be closer to the training data to be misclassified with high confidence. Nguyen et al. predicted that, were they to apply the same fooling strategy that was so successful against their target DCNNs to a generative model, high confidence synthetic images would look more like the target

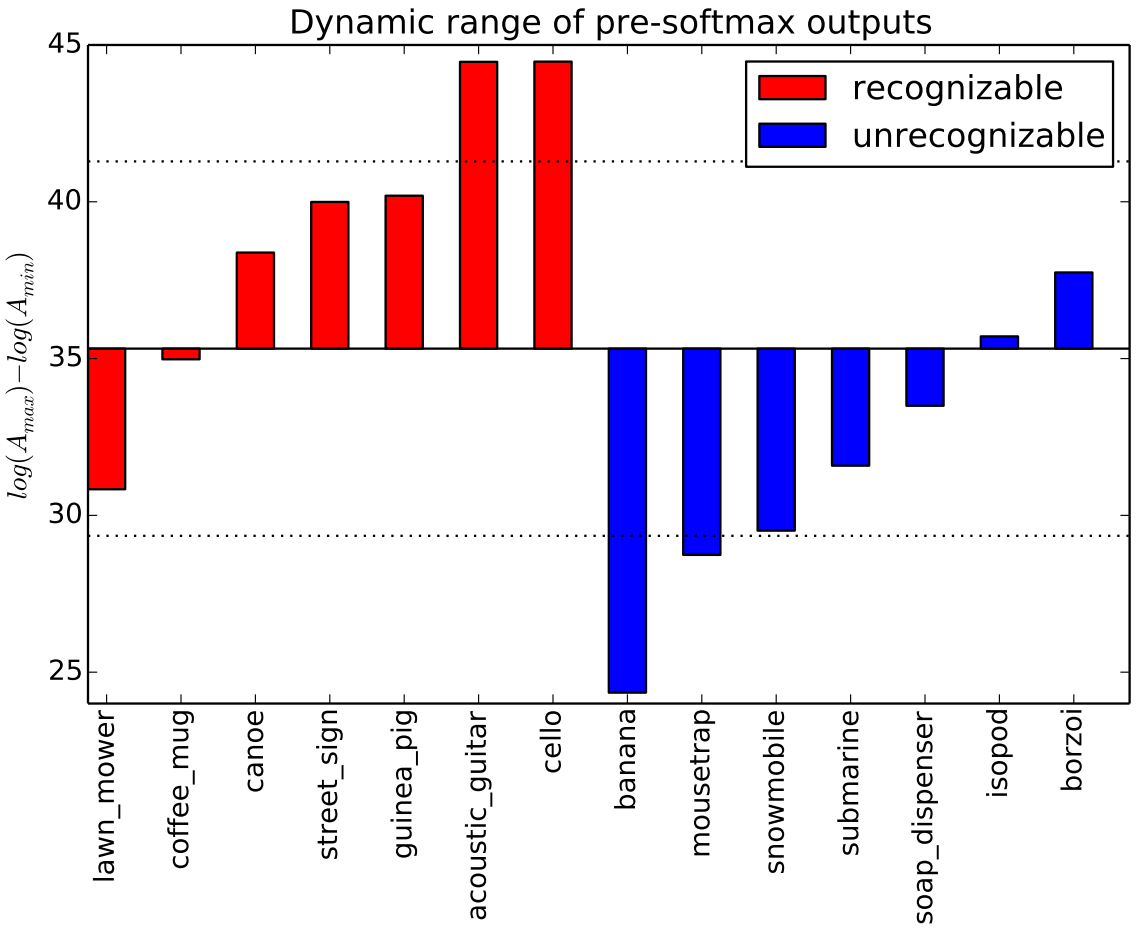


Figure 15: Dynamic range of pre-softmax activation values

Dynamic range, $log(A_{max}) - log(A_{min})$, plotted relative to the sample mean in pre-softmax activation of units within a sample of object types. 7 of the objects were universally recognizable from reduced-energy images by pilot study participants, and 7 were universally unrecognizable. The sample mean is 35.3 and the sample standard deviation is 6.00. The difference in dynamic range between the recognized and unrecognized categories provides evidence that the neural network uses a different classification strategy for each category.

class. Nguyen et al. identified the application of their synthetic image generation methods to generative models as a promising area of future work, because generative models lag behind discriminative DCNNs in image-benchmark performance and there are few generative models that scale to high-dimensional datasets of the complexity of ImageNet.

This work can be viewed as exploring a question related to the prediction of Nguyen et al. that generative models would be harder to fool: to what extent do discriminative DCNNs contain models of the categories they are trained to classify, that are even remotely similar to the models humans use? Unlike other explorations of fooling DCNNs, I constrained my adversarial search for high confidence images by starting with natural images that were correctly classified and removing signal energy from those images. I imposed constraints in the reduction algorithm to prevent it from modifying an image of one class until that image is misclassified with high confidence by the DCNN. The constraints do not, however, prevent extensive modification of an image while preserving high-confidence classification of the the modified image. It is therefore reasonable to suppose that my method restricts the modified images to fall within a smaller neighborhood of image space and remain closer, in some sense, to the correctly-classified natural images than an unconstrained image modification method would. If the feature space used by DCNNs is similar to that used by our own visual systems, the small-neighborhood restriction should produce visually-recognizable fooling images as Nguyen et al. [2015] predicted for generative models.

For certain classes of images, for example acoustic guitars, the modified images do look strikingly like the original class to a human observer. Other classes, for example Russian wolfhounds, bear no resemblance at all to the original category. In the experiment depicted in Figures 10 and 11, I found that iterated application of the signal-energy reduction algorithm to isolate and remove all high-confidence features from a target image produced low-confidence images that closely resemble the original class, and high confidence images formed by compositing the reduced-signal images that also closely resemble the target class. These results suggest that it is possible for DCNNs to learn models that are visually descriptive according to human intuition, but it is not clear whether they do so for all objects. Additionally, it seems that DCNNs are often able to classify an image based on a small subset of features that is not sufficient for humans to classify the image.

2.4.2 Class specificity of model descriptiveness

The example depicted in Figure 12 suggests that the recognizability to human observers of the reduced-energy images appears well correlated with the label of the original images, even when the images themselves are visually very different. This is surprising, as it suggests that the factors that lead to human-recognizable reduced-energy images are *properties of the neural network, not the images themselves.* The fact that some entire classes of objects tend to produce recognizable reduced-energy images and others tend to produce unrecognizable ones suggests that the essential features for classifying images are, for some types of objects, similar to the features that human visual systems depend on, and for other object types dissimilar.

This class-specific recognizability provides insight into a possible high-level strategy employed by the DCNN, in which classification of some object classes is based on prior rejection of other classes. In this strategy, the DCNN learns a strong descriptive model of some classes, where I approximate descriptive strength based on similarity of the features it relies on to the features that our own robust visual apparatus appears to rely on. Detection of other classes using the putative strategy depends on first ruling out classes having such strong descriptive models

within the network. The rules of the ILSVRC image classification contest reward correct classification of an image that is known to contain one of 1000 object types. Robust classification of some object classes is advantageous to classification of other object classes under these rules, because, having ruled out one of the objects that is robustly detected, the DCNN can use a more specialized set of features to distinguish between the remaining classes. Use of such specialized features combined with a process of ruling out objects detected via more generally-applicable features is qualitatively similar to cascades that Viola and Jones [2001] applied to face detection, in that more specialized features are only considered if their applicability has not been ruled out by the application of a more general set of features.

2.4.3 Ruling out alternatives: a DCNN strategy

I noted patterns in the pre-softmax activations of the DCNN that suggest that ruling out of alternatives plays a role in classificaion. The softmax layer provides one easily-measured mechanism by which the ruling-out strategy can be realized in the network. The softmax layer allows a unit with a very strong activation to completely overwhelm weaker units when the strong unit is active, yet not interfere with weaker units when it is inactive. Thus weaker units need only overwhelm other weaker units in order to achieve strong classification confidence, and they may be active when strong units are active without interfering with strong units. To put this in terms of the results shown in Figures 14 and 15, if the image contains a cello, it would not matter if the pre-softmax activation for banana is also relatively high compared to its average activation. Because the cello output is so much *louder* than the banana output, the softmax layer will still output high confidence in cello. That means the banana detector is free to produce spurious detections as long as a loud category is present; the spurious activity will not affect the classification very much.

If loud units correspond to classes detected via a visually-robust set of features, and quiet units correspond to classes detected via brittle, specialized features, then the strategy naturally emerges where robustly-detected classes are ruled out in order to facilitate high-confidence detection of contingent classes. The existence of trend like that depicted in Figure 15 toward louder activation in classes which are more recognizable from their reduced-energy images is encouraging evidence that ruling out plays a role in DCNN classification.

Figure 11 illustrates a surprising observation about the features DCNNs use to classify images. In the experiment in which I iteratively removed signal energy corresponding to the reduced-energy image until a marked drop in classifier confidence occurred, it appears that the image generated by accumulating all of the reduced signal energy resulting from each iteration was a nearly complete reconstruction of the object of interest, in the case shown in Figure 11 a hare, with nearly all of the background removed. Each iterative run of the signal-energy reduction algorithm was forced to produce a distinct reduced-energy image, because the signal isolated by the previous iteration was removed from the image. In the case of the hare in Figure 11, none of the reduced-energy images is recognizable alone despite that they are very recognizable in aggregate, suggesting that the DCNN is selective for a relevant set of features in images even though it does not require all of them to classify with high confidence.

2.4.4 Implications

A training strategy in which the network is retrained on residual images like those in Figure 10 could be beneficial in capturing more subtle features. Also of interest, a detection strategy in which the minimal set of features responsible for a given high-confidence detection are suppressed and then the network is re-applied to the image could produce a more robust classifier if the detections are stable over several iterations. This kind of detection strategy is especially interesting because it uses feedback to make sure that the classification is supported by more than one minimal set of image features. Although the signal-energy reduction algorithm I used in this work is likely too computationally expensive for use in a real-time object detection, the principle of using feedback to make sure objects are still recognizable as salient features are removed may prove to be a powerful strategy to defeat adversarial fooling and brittle phenomena in general.

The results emphasize the need to choose good problems to use as testing grounds in visual intelligence, as I argued in Section 1.3. The DCNN in these experiments appears to have learned a kind of computational shortcut in which it rules out certain categories so that it can distinguish among the others using more specialized features. The strategy works well in that it leads to good test performance on the 1000-way classification task. The reduced-energy images clearly show that, for certain image classes, the model does not capture the type of compositional understanding of the class that is required by robust visual intelligence. Placing additional constraints on a network to force it to learn more relevant representations is a subject revisited in Chapter 5.

The alignment hypothesis anticipates that requiring the network to interpret the images in the context of multimodal perception would place meaningful constraints on the image features the network would learn. For example, if the decision to output *dog* were based on a multimodal representation that included a depth map in addition to pixel colors, and this depth map had to be estimated from the image if it was not given as an input, then I would expect that the reduced-energy images would have to contain at least enough information to capture the aspects of the object's geometry that make it a dog. This experiment remains a goal of future work.

2.5 Contributions

My main contributions in this chapter are as follows.

- I developed a methodology for evaluating similarity between the features most crucial to classification via a DCNN, and the features used by our own visual system, in which I remove content from natural images without reducing classification confidence by the DCNN, and evaluate the recognizability to humans of the resulting modified images.
- I implemented an algorithm based on randomized search that reduces signal energy of natural images to local minima, preserving DCNN classification confidence.
- Using the pre-trained AlexNet model of Krizhevsky et al. [2012] provided by the Caffe software package developed by Jia et al. [2014], I applied my algorithm to 142 high-confidence images.
- Through a pilot experiment with 5 participants, I identified preliminary evidence of a pattern in which certain entire *classes* of images, such as acoustic guitars, produce recognizable

reduced-energy images via application of my algorithm, whereas other image classes such as Russian wolfhounds produce unrecognizable reduced-energy images.

- I advanced a hypothesis to account for the class specificity of recognizability of the modified images by human observers. I hypothesize that the DCNN rules out robustly detected classes before applying specialized features to detect the remaining classes. I found evidence supporting my hypothesis though analysis of dynamic range in pre-softmax activations in the DCNN.
- Through iterative application of my signal-energy reduction algorithm, I produced an image which is misclassified yet retains the appearance of the original class. I predict that applying the signal-energy reduction algorithm in such a way could yield useful synthetic training data. Likewise, ensuring that classification remains stable over several iterations of signal-energy reduction may provide a route to strengthening robustness of classifiers.

3 Foundational Work in Constraint Propagation

In this chapter you learn how my work with visual propagation differs from pioneering work in related areas. I introduce distinctions between types of propagation, and make observations about how propagation naturally arises in certain image-centric computations. I briefly discuss exploratory work that led to my work on propagators that is the subject of Chapter 4. In the exploratory work, I extended the procedure of Waltz [1972] using a probabilistic graphical model.

3.1 Introduction

My work on implementing models of visual intelligence as processes that align sensory data with expectation relies on the methodology of propagation, and specifically a propagation programming methodology inspired by the work of Sussman and Radul [2009]. In this chapter, I make some observations about a potential reason why propagation is a good strategy to apply to problems in vision, and make several distinctions that differentiate propagation methods. I review notable examples of propagation applied to problems in vision and contrast the examples with my own work with visual propagators. I then review aspects of Sussman and Radul's propagator architecture, which forms the inspirational groundwork for the propagation systems I use in my own work, identifying the main similarities and differences between that groundwork and my own contribution.

3.2 Observations about constraint propagation systems

A powerful idea that emerges in many successful computational models of vision processes is the idea of embedding computations in a 2-dimensional image-like space. Analogous to the retinotopic computations observed in natural vision systems, computations that operate on feature maps that are 2-dimensional and homeomorphic with the original image are able to take advantage of the property of natural images that nearby observations are closely correlated. Hierarchical, adjacency-preserving representations have prevailed from pioneering work on primal sketches and 2½-D sketches (Marr [2010]) to state-of-the-art convolutional neural networks. Computations structured to operate on such representations can be thought of as implementing a type of implicit constraint propagation, by performing local computations on 2D regions of gradually increasing scope (or receptive field), and achieving global consistency via constrained interactions at the boundaries of the local regions. In this sense, a variant of constraint propagation could be interpreted as underlying most work in mid- and high-level vision. I limit my survey of foundational work in visual propagation to techniques that are conspicuously constraint driven, rather than techniques in which the constraint propagation emerges as a byproduct of computational structure. Specifically, I consider techniques that explicitly model the constraints under consideration. Explicit constraint propagators operate by refining a single description via iterative application of the same set of constraints. A deep convolutional neural network satisfies neither the explicit-constraint criterion nor the single-description criterion, because in the neural network the constraints are emergent properties rather than explicitly-represented ones, and instead of applying its constraints repeatedly to the same evolving description, deep convolutional neural networks build up a hierarchy of descriptions, each produced by one layer and used by a subsequent layer. A final, important note on scope and terminology is that I use the

term *constraint propagation* as it is used by Winston [1992]. Specifically, I do not limit its scope to logical constraints of satisfiability problems as some authors do, but include the somewhat idiomatic *numeric constraint propagation* family of propagation procedures as well.

Among representations that explicitly model constraints, there are several distinctions of interest. One distinction is between numeric constraint propagators and symbolic constraint propagators (Winston [1992]). Numeric constraint propagators operate on variables containing information about discrete or continuous scalar or vector values, whereas symbolic constraint propagators operate on variables containing symbolic descriptions. The symbolic representation space may be finite, as in the case of the edge-label possibilities in Waltz's procedure (Waltz [1972]) or infinite, for example, the space of natural-language sentences. Another distinguishing factor among constraint propagator systems is whether the systems always honor consistency while increasing specificity under a set of constraints, or, alternatively, approach consistency as determined by the constraints while remaining maximally specific.

In consistency-preserving systems, such as Waltz's procedure, variables retain collections of all currently-known permissible values at a given stage of constraint propagation. When constraints are applied, they prune values from the collections of permissible values, making the collections more specific. The dual of the consistency-preserving pattern is the specificity-preserving pattern. These systems start with some, possibly random, initialization of variables that is maximally specific, as opposed to a collection of possibilities. By a process of iterated application of constraints, or *relaxation*, the values assigned to the variables move toward consistency as determined by the constraints. In such relaxation-based architectures, there is a danger that the state configuration may diverge or oscillate. In architectures that preserve consistency, a danger exists that the system may fail to make progress in eliminating possibilities. This danger of not reaching full specificity even when solutions exist reveals another important distinguishing factor among constraint propagators: whether the systems are pure constraint propagators, solving constrained problems using propagation alone, or whether the systems employ explicit search when they get stuck in configurations from which propagation can make no progress.

Other factors of variability exist in the world of constraint-based representations. For example, certain constrained problems lend themselves to solution via well established optimization methods. Problems of interest to me, however, tend to have many heterogeneous constraints that would make framing the problems in such a way that they can be solved by established optimization methods at best unwieldy to specify, and more likely intractable. Together, the distinctions between numeric and symbolic, between consistency-preserving and specificity-preserving, and between pure propagation and propagation plus search represent important considerations when designing propagator systems. In my work, I focused on systems that do not use explicit search. The urgency of many of the goals that natural vision has adapted to serve clearly precludes extensive search in some cases. My interest is to address these cases rather than high-level visual tasks where explicit search may become a relevant strategy. In the work described in Chapter 4, I found numeric constraints to be more prevalent and informative, and specificity-preserving methodologies that are implemented via relaxation avoid specific problems with scalability and brittleness of logical absolutes (for in-depth discussion, refer to Section 4.4).

3.3 Applications of constraint propagation in vision

In this section I present and contrast with my work several notable early examples of constraint propagation applied to vision problems: Horn's procedure for deriving shape from shading, Waltz's procedure for determining 3D structure from line drawings, and Hinton's work on using relaxation to find globally consistent interpretations of figures comprised of overlapping rectangles.

3.3.1 Shape from shading

A method developed by Horn (Horn and Brooks [1989]) derives surface normals of a depicted object in a single monochromatic image. The method makes use of two constraints: that nearby pixels ought to have smoothly varying surface normals in most cases, and that the brightness of a pixel specifies a set of allowable surface normals on the corresponding isobrightness curve of the reflectance map, for a given surface material and lighting configuration. The constraint propagation algorithm is of the numeric, specificity-preserving type according to the breakdown of constraint-propagation methods presented in Section 3.2, in which all variables contain continuously-variable vector values, and the variables maintain maximum specificity at all times, gradually approaching consistency under the constraints. The surface normals for the pixels at the edge boundaries of objects are known and left unmodified by the constraint propagation algorithm. The remaining pixels approach consistency via relaxation under the brightness and smoothness constraints.

My work with propagation has several conceptual differences from this pioneering work on estimating shape from shading. The most conspicuous difference is that instead of operating with a small number of constraints arising from the physical interaction of light and surfaces, my work on propagators uses many, heterogeneous constraints arising from many domains. An example of a constraint that my system exploits is that people's heights fall within a predictable range of values. Rather than attempt to guarantee performance attributes such as convergence conditions, I sought to build a framework that facilitates integration and evaluation of new constraints so that performance can be measured empirically.

3.3.2 Waltz's 3D-labeling procedure

Waltz developed a method of using constraint propagation to label the edges in line drawings representing arrangements of objects with piecewise-planar surfaces (Waltz [1972]). Line segments in such drawings can represent different features of physical object arrangements, for example cracks between blocks, concave boundaries, convex boundaries, shadow edges, and occlusions or bounding limits. Junctions are defined as points where line segments meet, and are distinguished by the number of intersecting line segments and the angles between each pair of segments in the junction. Waltz identified a finite taxonomy of junction classes in drawings originating from world states with certain restrictions, for example, the restriction that only a specified number of faces can share a vertex, and the restriction that junction types must be invariant to small changes in viewpoint.

Constraints on the ways in which line-segment types can participate in particular classes of junctions place limitations on the sets of allowable line-segment types. The constraint propagation process first prunes the allowable line-segment types based on the junction pairs that each

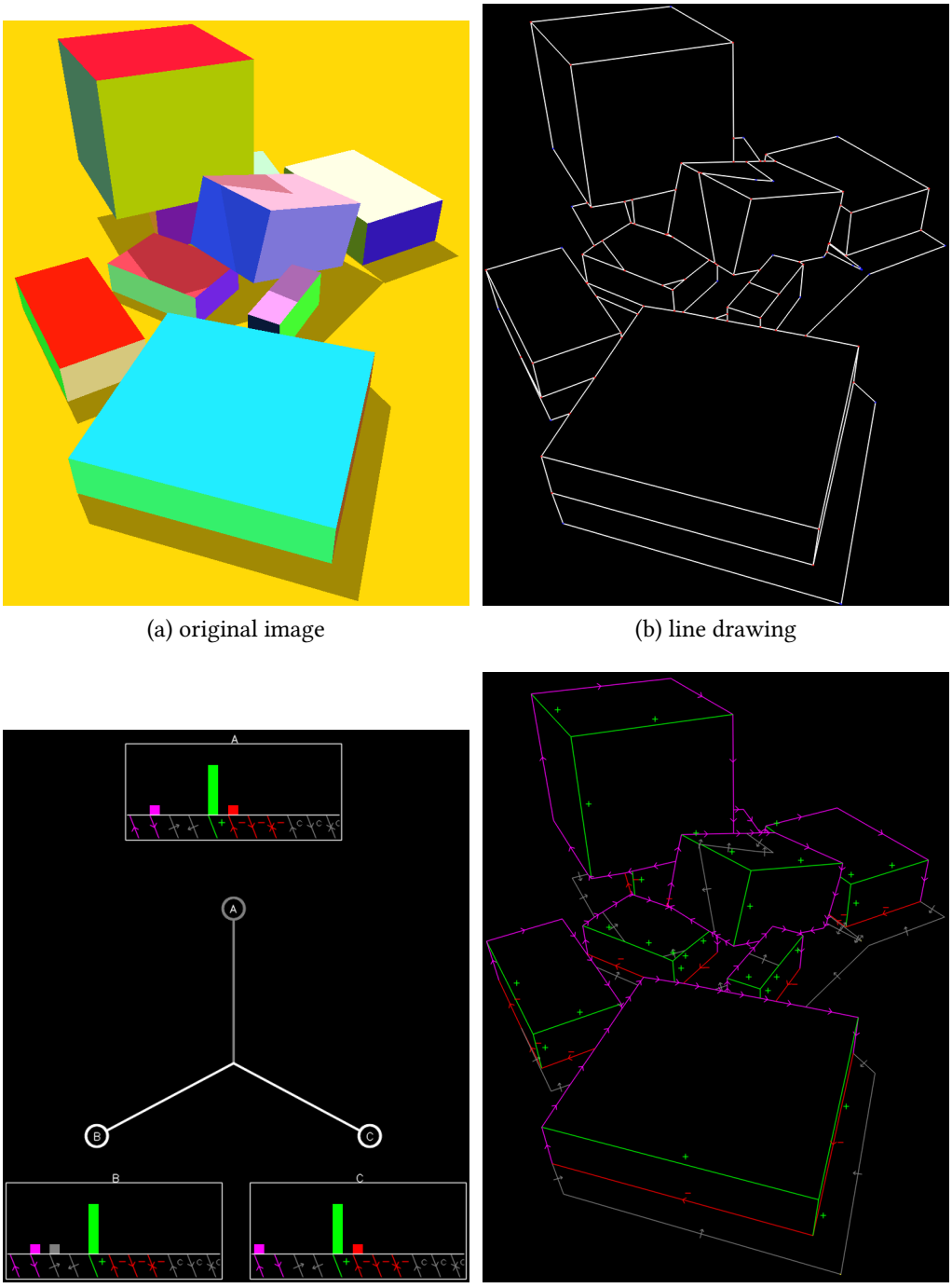
line segment connects with. Then, the process prunes the sets of allowable junction configurations in each junction based on the updated sets of allowed types of each line connected to the junction. Iterating the process until it reaches a steady state can efficiently rule out many internally-inconsistent interpretations of the line drawing, and in practice often converges to a single global interpretation in which all line segments retain exactly one permissible label. Waltz's procedure is an example of symbolic, consistency-preserving constraint propagation according to the breakdown of constraint-propagation methods presented in Section 3.2, in which variables contain sets of all possible symbolic assignments not known to be inconsistent, at a given iteration of constraint propagation.

Initial work on expanding Waltz's procedure In exploratory work that led indirectly to the work described in this thesis, I implemented a probabilistic variant of Waltz's procedure. Rather than use a predetermined library of junction classes, such as arrow-shaped or T-shaped junctions, I used a finer-grained family of junction types determined by the number of edges and the quantized angles between them. I developed a canonical representation for these junction types to facilitate aligning them with one another. Using a physics simulation, I measured statistics from many randomly-generated world states to estimate the frequency of each junction type. The statistics were stored in a tensor representation. One rank-N tensor records the statistics over all possible N - edge junctions (tensor indices are $\theta_1, ..., \theta_{N-1}, J$, where θ_ℓ is a quantized angle and J is the index of a unique junction labeling). The tensor associates quantized angle measurements with a marginal probability distribution over all possible edge labelings. To label a new line drawing, I converted the line drawing to a Markov random field, in which edges in the original line drawing become the random-variable nodes in the MRF, each junction becomes a clique, and the empirically-measured junction statistics determine the factor potentials of the cliques. Inference is then a straightforward application of loopy belief propagation. Elements of the process are illustrated in Figure 16.

The main difference between my work described in Chapter 4 and Waltz's procedure is that my work focused on the scenario where there are many heterogeneous constraints governing local interactions, not all of which are strong logical constraints. For example, the observation that humans are usually physically supported by other objects is not universal: a person could be falling through the air or attached to an invisible wire. The constraint is nevertheless useful as a guiding principle when interpreting scenes in which humans are present. Unlike my initial work on expanding Waltz's procedure using graphical models, in my subsequent work with propagation I sought to find explicit constraints: those that can be described in terms of principled analysis, rather than those that rely purely on statistical analysis.

3.3.3 Hinton's work on relaxation

A notable early example of constraint propagation under uncertainty is Hinton's puppet-finding program (Hinton [1978]). Hinton's puppet-finding program interprets the semantics of 2D drawings consisting of overlapping rectangles. The inputs to the program are lists of rectangles specified as coordinates of their respective corners, and instructions to guide interpretation. Each instruction has an associated numerical importance value, expressed as an unbounded real number, indicating how important it is to obey the corresponding instruction. The outputs are labels for each rectangle and joint, specifying which part of a marionette-like puppet each represents.



(a) original image

(b) line drawing

(c) edge statistics

(d) line labels

Figure 16: Statistical reframing of Waltz' procedure

In my implementation of a version of Waltz's procedure extended to probabilistic inference, an image of blocks (a) generated randomly in a simulation is automatically processed into a line drawing (b). Statistics measured from many simulated arrangements are visualized for a specific 3-edge junction in (c). The statistics enable reconstruction of the edge labels (d) via loopy belief propagation, a type of numeric constraint propagation applied to probability values.

For example, the rectangles may represent head, neck, thigh, etc., and the joints may represent knee, elbow, ankle, etc. The puppet-finding program works by forming three interlinked networks of data structures representing rectangles, potential label hypotheses, and suppositions. The supposition network contains numerical values corresponding to the initial importance, if specified, of each hypothesis, and a supposition value for each hypothesis that arbitrarily starts at zero. The supposition network also implements constraints, which a relaxation algorithm uses to iteratively update supposition values of a given node based on the supposition values of a node's neighbors.

The hypothesis network in Hinton's implementation is initialized with locally-plausible interpretations of joints and rectangles when such local interpretations exist. Built-in rules are used to identify locally-plausible interpretations. For example, a rectangle that overlaps one other rectangle and is wider than that rectangle is interpreted as a candidate head. After initialization, the hypothesis network permits nuclei of highly-constrained interpretation to propagate, via the supposition network, in a manner that is evocative of belief propagation in graphical models, albeit without the requirement that the supposition values being relaxed have a defined meaning in terms of a probability distribution.

This early work by Hinton anticipated key aspects of the problem of modeling robust visual intelligence: that vision systems need to formulate tentative hypotheses, that such systems need to be able to find consistent sets of hypotheses, that constraints expressing the relationships between hypotheses should be explicitly represented, and that a method that achieves constraint propagation via relaxation is desirable. Such promising directions have been left largely unexplored by contemporary vision research, for example, research involving deep neural networks. In direct contrast with these early observations by Hinton, deep neural networks do not formulate tentative hypotheses in a meaningful way when they are feed-forward, because being feed-forward precludes the ability to revise hypotheses. They do not find consistent sets of interpretations, but rather operate by finding interpretations that are similar to training examples, having no way to evaluate the internal consistency of those interpretations. They do not explicitly represent constraints; all constraints are instead emergent properties of the learned representation. Rather than operating by relaxation or constraint propagation, most neural networks used in vision are feed-forward and therefore may only operate on hierarchical descriptions. Subsequent work by Hinton and others on deep Boltzmann machines and other neural network architectures that can perform feedback may provide a route to hypothesis generation and testing via relaxation, but still fall short of the ability to explicitly represent constraints. My work aims to address both of these issues.

3.4 The propagator architecture

The architecture I describe in Chapter 4 is inspired by pioneering work on a propagator-based programming system by Sussman and Steele [1980] and further developed by Sussman and Radul [2009] and Radul [2009]. This propagator architecture addresses shortcomings of expression evaluation, which is the foundation of almost all modern programming systems. In particular, the propagator architecture permits components to operate independently on partial results of computations, rather than having to wait for a computation stage to be completed before using its results, as is the case with expression evaluation. A program implemented in this propagator architecture comprises a set of stateless propagator machines, and a set of cells that store

and manage state. The sets of propagators and cells connect with each other to form a bipartite computation graph in which edges represent paths along which information can flow in either direction. Cells store information about values, rather than values themselves. For example, an empty cell that is capable of representing real values would be interpreted as potentially containing any real number. A propagator connected to such a cell could push an update that specifies that the value of the real number in question is greater than 0. The cell then notifies the system that its domain has been reduced, and sometime later its connected propagators are applied to spread the effects of the domain reduction throughout the graph. Bidirectional information flow allows propagators to enforce relations among connected cells, rather than limiting them to enforcing simple functions that map input to output. The order of execution of propagators is not specified in the description of a system of propagators and is expected to not have an effect on the values of cells at quiescence, after all updates have been processed and no further information can be accumulated in any cell. This property allows flexibility in a propagator system's control flow: the scheduler may opportunistically execute propagators, and it should have no effect on the quiescent value of cells whether propagators are simulated by a uniprocessor or whether they are all parallel, independent machines.

Cells track and monitor their updates. The tracking enables analysis of data provenance, by showing which supplied data support a given conclusion through which chain of deductions. The monitoring allows detection of conflicting updates: for example, if one update specifies that a real-valued cell's value must be greater than 0, while another specifies that the value is less than -1, then the cell will signal that an error has occurred. The propagator architecture allows for flexibility in the way such errors are handled. The system may signal an error and then either halt or enter an error-handling context in a way that is familiar in most programming environments. Another option is based on dependency-directed backtracking, (Stallman and Sussman [1977], Zabih [1988]), with which the system optimizes the search for mutually-consistent sets of assignments based on remembering which minimal subsets of assignments are mutually inconsistent. Dependency-directed backtracking empowers truth-maintenance systems (TMS) (McAllester [1978]) which allow propagator systems to reason about several internally-consistent but mutually-inconsistent worldviews simultaneously.

The propagator architecture of Sussman and Radul [2009] was an appealing candidate for my work with alignment in vision, because it provides a principled and elegant way to express how the components of a complex system interact via their shared states and representations, without needing to compromise on such principles or elegance in order for the local interactions to give rise to complex behavior. The architecture thus invites the opportunity for surprise, rather than bewilderment, about any complex behavior that does emerge from the local interactions among semantically well-characterized components. I believe that it will be by way of such surprises that we will discover *explanatory theories* of our own intelligence, rather than mere statistical characterization of our intelligence's inputs and outputs.

In Chapter 4 I explore the implementation of a simple vision system that uses propagators. Development of the system led to insights about difficulties in applying the architecture to some problems of interest in vision, and eventually to ways to integrate propagator-like components with neural networks, as I present in Chapter 5.

3.5 Summary

In this chapter, I reviewed a set of distinctions among propagation methods: whether they are symbolic or numeric, whether they are consistency-preserving or specificity-preserving, and whether they use pure propagation or propagation combined with explicit search. I determined that the architecture of most interest in this work is the numeric, specificity-preserving type that does not use explicit search. I discussed three early uses of propagation in vision: Horn's shape from shading, Waltz's edge-labeling procedure, and Hinton's work on using relaxation algorithms to find puppets. I introduced the inspiration for the propagator architecture that I used, and motivated the architectural choice.

4 Processing a Scene with Propagators

In this chapter you learn about my work on implementing vision systems using an alignment-driven methodology and a propagator architecture. I present the implementation of a system that uses propagators to track pedestrians and uses the tracking along with constraints to infer geometric properties of the scene like ground-plane position and occluder locations. Figure 17 depicts example output of the system. I also present an in-depth discussion of the system's limitations in order to break free of those limitations.

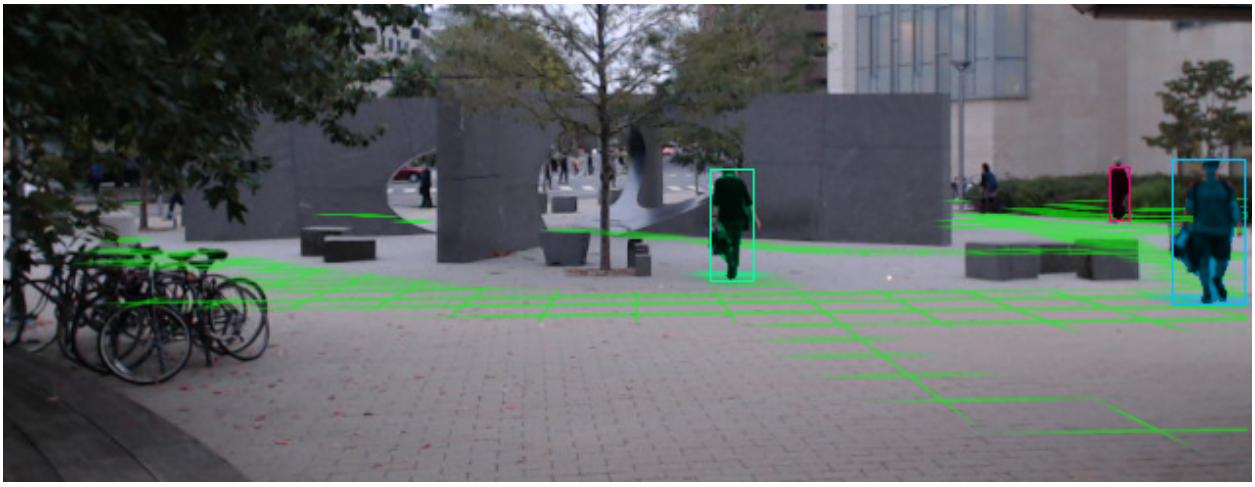


Figure 17: Example output of propagator system

One ability of the propagator system that is the subject of this chapter is to use pedestrian tracks, along with knowledge about average adult height, to estimate the parameters of the ground plane. The ground plane is shown in green where supported by observed tracks. Grid squares are 1m.

4.1 Introduction

To make progress toward providing story-understanding systems and general problem solvers with the ability to visually perceive and understand events, I believe we need to build ways to infer the geometric constraints of 3D scenes in which events occur. To realize the potential of pervasive alignment of partial information to give rise to robust perception, as discussed in Chapter 1, it is essential that high-level knowledge derived from story-level processing of events be able to direct and influence the way that low-level data are interpreted. The state of the art in AI reasoning systems with a visual input is illustrated in Figure 18 (top). Perception is modeled as a multi-stage feed-forward process in which higher stages model the uncertainty of the lower stages supplying the information, in order to resolve ambiguity. The proposed alignment-based architecture is depicted in Figure 18 (bottom). In this type of architecture, the perceptual apparatus accepts feedback from the story understanding system. This feedback contributes to an evolving understanding of scene geometry, allowing high-level information from the story processor, such as known dimensions of objects, to have direct bearing on how low-level scene characteristics, such as depth maps of supports and occluders, are generated.

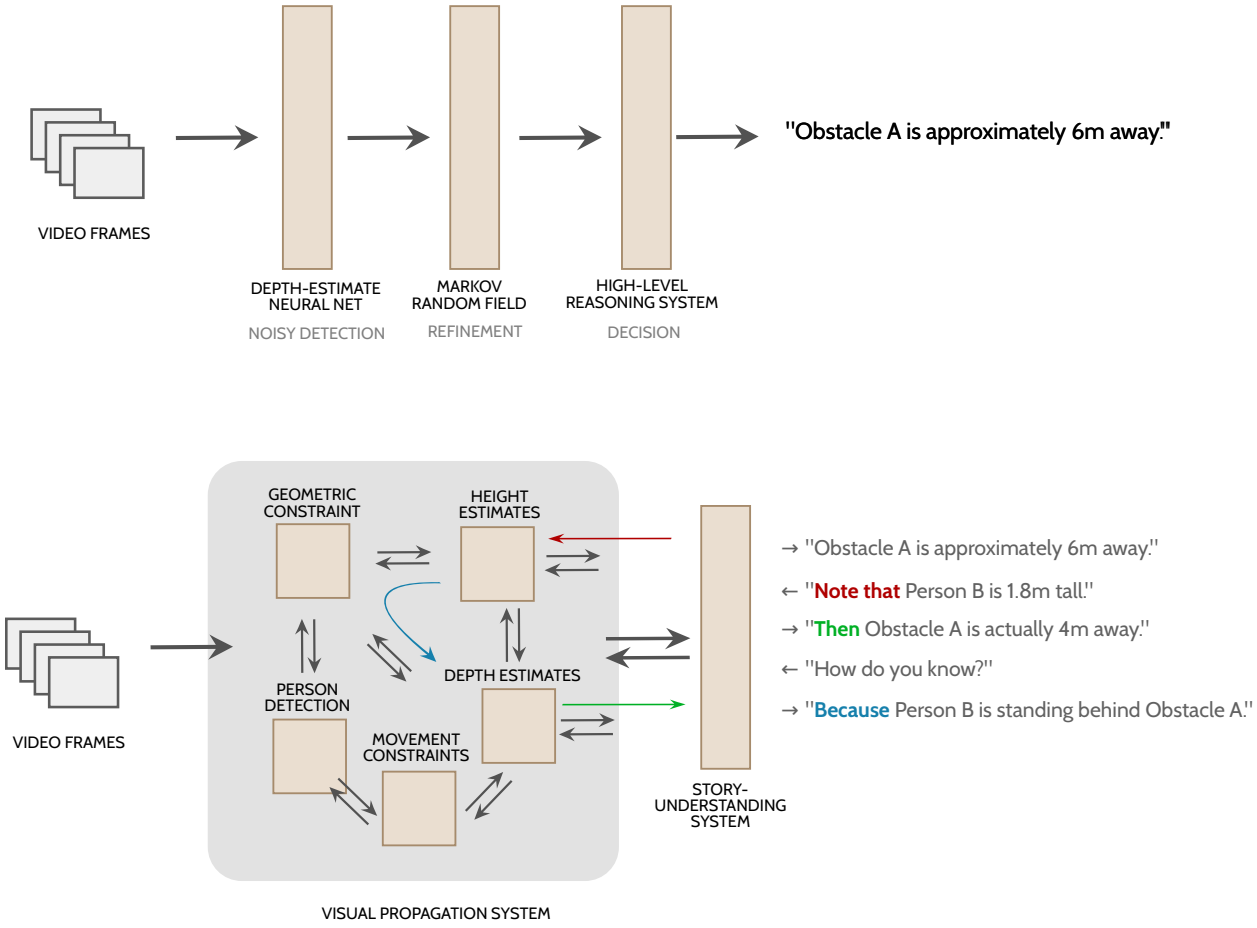


Figure 18: Comparison of propagator and pipeline approaches

The feed-forward approach (top) cannot directly assimilate high-level knowledge in the interpretation of low-level stimuli, relying instead on a hierarchy of probabilistic graphical models to resolve uncertainty. The visual propagator architecture (bottom) allows integration of high-level information, e.g. height of a known actor, into processing of scene-level constraints such as depth maps of occlusion boundaries. Furthermore, the visual propagator architecture can offer justification for its output.

In this chapter I present a case study in pursuit of alignment using the propagator mechanism introduced in Chapter 1 and expanded on in Chapter 3. I introduce a vision system that I implemented based on that propagator mechanism. The design of this system, and the insights revealed by the implementation, represent a step toward realizing the capability depicted in Figure 18. Given video taken from a stationary vantage point in an outdoor urban scene, the system can combine information from tracked objects with externally-supplied information, of the type that a story-understanding system could provide, to make inferences about scene geometry. The system can then transfer the resulting knowledge about scene geometry to new foreground objects, providing justification for its inferences in terms of both its own observations, and the externally-supplied knowledge.

The vision system presented here serves as a simplest complex example: an example with just enough complexity to demonstrate how the propagator system's components function together in a real-world scenario. The design goal of the system was to achieve proof of concept and to

motivate and inform further development using a propagator-driven design methodology, not to outperform state-of-the-art computer vision systems on benchmarks. As such, my emphasis is on describing how the system works and on demonstrating its output on some examples, rather than quantitatively evaluating its performance. In some ways, the system did not realize the design goal. In particular, it was problematic to achieve pervasive alignment, which is required by the alignment hypothesis that is the central theme of this work. The effort did yield extensive insight into such problems, and how to address them.

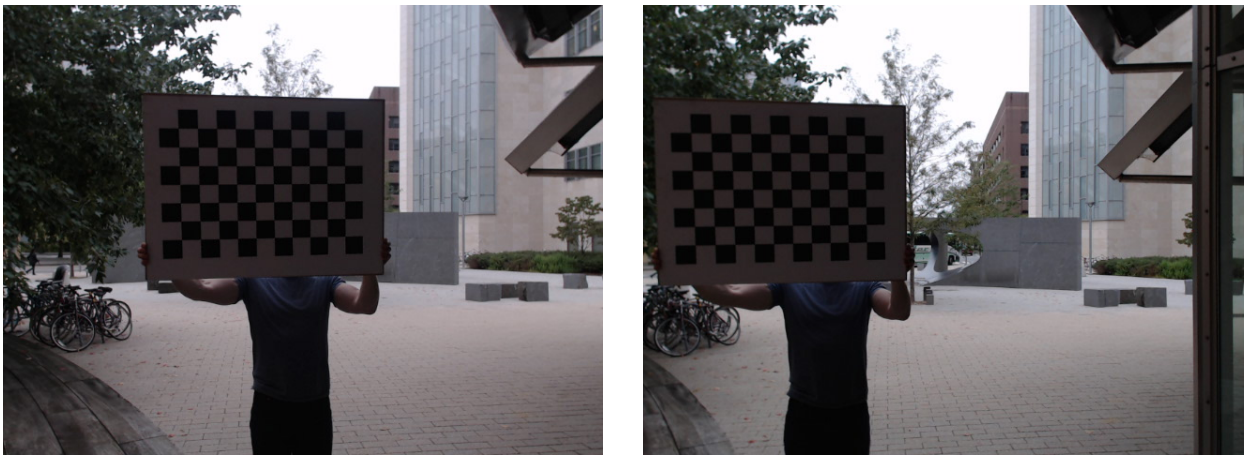
The rest of this chapter is organized as follows. I describe the setup in Section 4.2, including the type of data collected and the additional inputs and outputs of the system. I present details of the implementation in Section 4.3 with emphasis on aspects that I believe are widely applicable and present examples of the system's performance. In Section 4.4 I discuss lessons learned from observed shortcomings of the approach, and ways to take the concept to the next level.

4.2 Experimental setup

Although the methods described in this chapter use input from only one camera, I collected video data simultaneously from two fixed cameras, using the fixture shown in Figure 19. Processing the video from the additional camera produced stereo disparity maps to evaluate the accuracy of geometric measurements. An example stereo image pair from the cameras is shown in Figure 20. The two cameras were 0.5 meters apart, and positioned so that their respective optical axes and x and y axes were parallel. Video frames were captured at 30FPS, in RGB, and with a resolution of 640x480 pixels per frame. Additionally, the camera positioning ensured that the image plane was vertical, simplifying the relationship between image height and object height.

Figure 19: Data-collection apparatus

The cameras used for data collection were placed 0.5m apart with all axes parallel. The cameras are positioned so that the images are co-planar and the y axes in camera coordinate space are vertically oriented. The fixture adheres to the outside of the MIT Stata Center using suction cups.



(a) Left Camera Image

(b) Right Camera Image

Figure 20: Stereo images from the camera array
The left and right images from the cameras illustrate the large stereo disparity. The checkerboard pattern in these images served as a calibration target to estimate the intrinsic matrices for the cameras.

The system produces bounding rectangles and unique identifiers for tracked objects, as illustrated later in the chapter in Figures 29 and 30. Additionally, the system produces locations where pedestrians commonly enter and exit the scene, as depicted in Figure 31. The system also approximates distances from the focal plane to foreground objects and background occlusion boundaries when available, along with detailed support information that links the inferences about scene geometry to the specific observations of tracked foreground objects that support each inference.

4.3 Implementation details

4.3.1 Abstractions

I used many of the same fundamental abstractions used by Radul [2009]. Propagators are stateless machines that respond to changes in their connected cells. Cells store state, and notify their connected propagators when that state changes. The cells should not forget information; propagators must issue only updates that increase the specificity of cells' states. A scheduler oversees execution of the propagator graph by executing all propagators attached to a cell whenever the cell accumulates new information. Figures 21 and 22 depict summaries of the most essential classes and interfaces that define my propagator system.

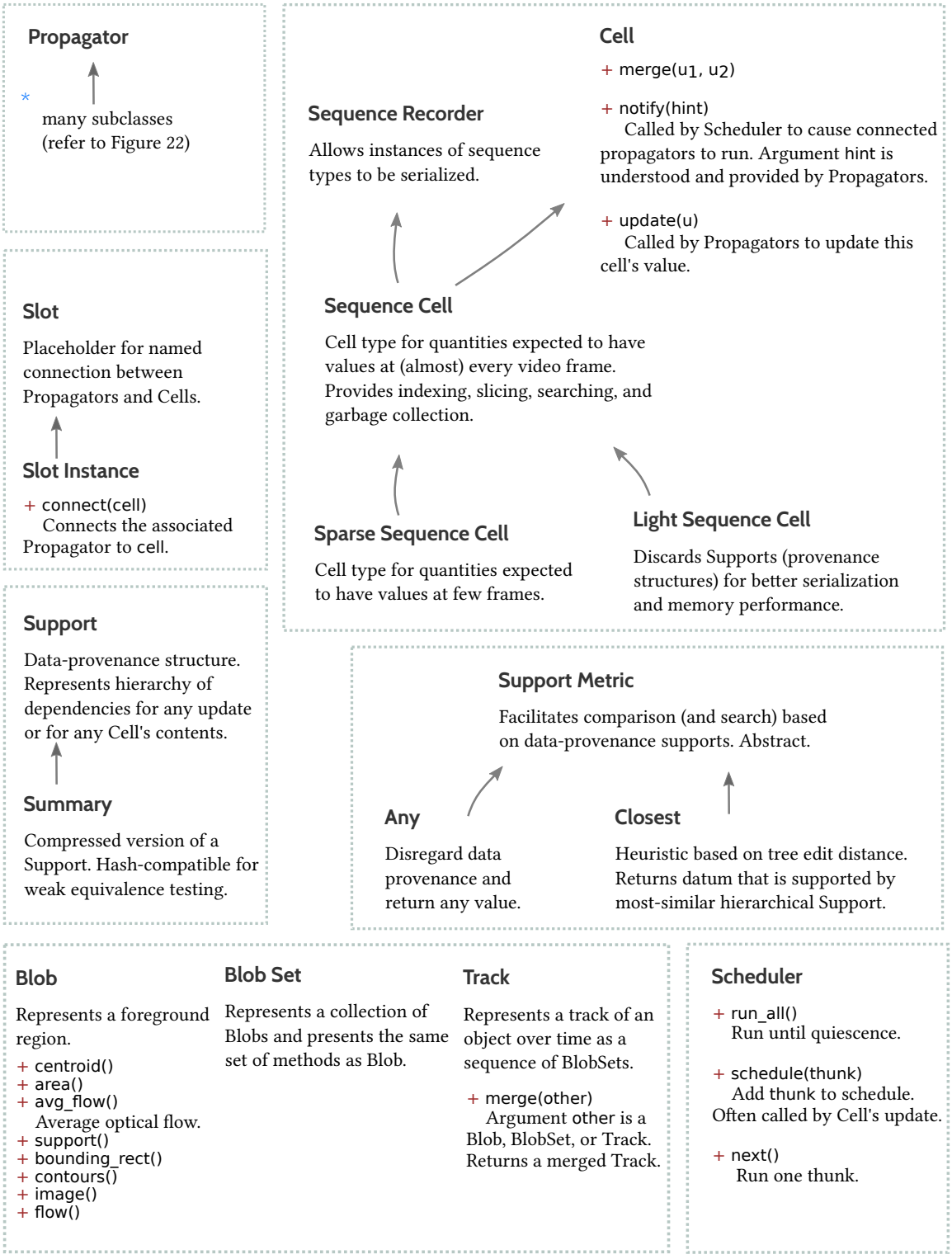


Figure 21: Inheritance and interface summary for low-level propagator system

In this sample of components within the part of the propagator system that deals with low-level visual processing, arrows indicate the `inherits-from` relation. Where appropriate, a representative sample of methods are shown. Many components, listed in Figure 22, inherit from `Propagator` and several classes (not shown) specialize `Cell`.

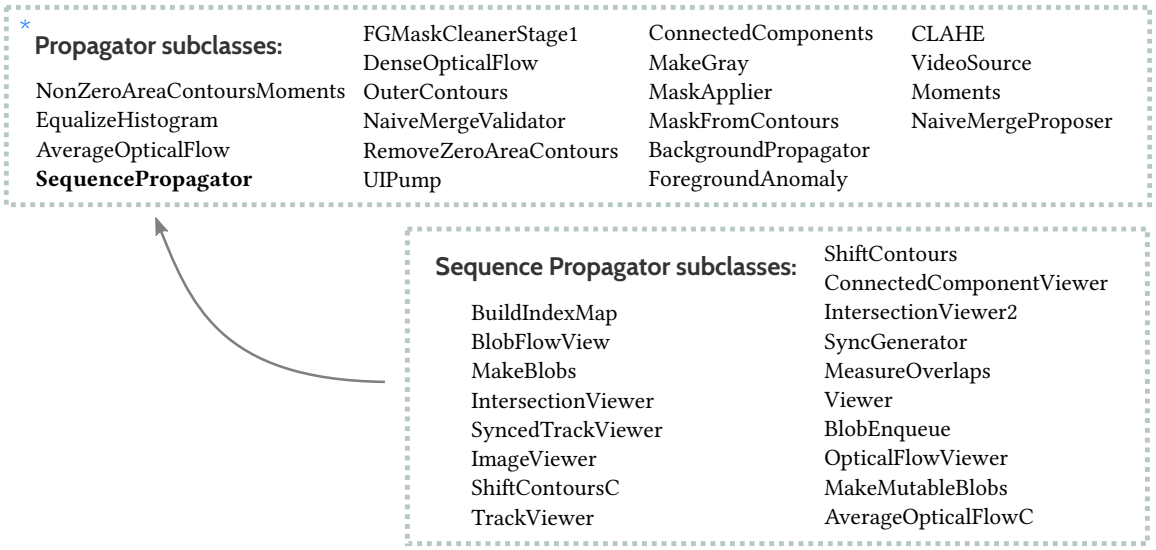


Figure 22: Propagator subclasses

The subclass hierarchy of `Propagator` used in this implementation contains 37 classes. `SequencePropagator` provides conveniences for propagators that interact with sequential data.

The system contains 37 propagator subclasses, with functions ranging from simple tasks such as sourcing frames from cameras or video files (`VideoSource`), to specialized tasks such as finding all connected components in a mask (`ConnectedComponents`), updating a mixture-of-Gaussians background model (`BackgroundPropagator`), and performing contrast-limited adaptive histogram equalization (`CLAHE`). The system contains 6 subclasses of `Cell`: `BackgroundModel` which specializes in mixture-of-Gaussians background models, `TrackCell` for storing tracks of objects, `SequenceCell` for storing sequences of descriptions, `SparseSequenceCell` for storing sequences that are not expected to have a specialized value at every sequence number, and `LightSequenceCell` which ignores provenance objects to reduce resource use. Out of necessity, sequential-valued cells must forget state that is no longer being used by the propagator system to avoid running out of memory. A sophisticated garbage collector could use support information to prune sequences, but this implementation uses a simple time horizon to remove old cell contents. Additionally there are `Support` classes which represent provenance of updates and cell values, `SupportMetric` types that facilitate comparison of updates or cell values base on their provenance hierarchy, and various infrastructure classes: `Slot` objects are markers used by the framework to record a `Propagator` subclass's availabile connection slots, and `SequenceRecorder` objects record a network's activity for playback later, so that computation that is resource constrained can be split up and processed in stages. The `Track` class and its variants are used for object tracking.

To demonstrate how the abstractions simplify the implementation, Listing 1 contains an example of a full implementation of a propagator. In this case, you see `AverageOpticalFlow`, a propagator that integrates optical flow inside of foreground contours to produce approximate average translation vectors for each contour. The CPU-intensive part is relegated to an external library that I implemented in C++.

```
class AverageOpticalFlow(SequencePropagator):
    """
    computes the average optical flow within each contour
    """
    #pre-computed areas of contours:
    in_areas = Slot()
    #lists of contours:
    in_contours = Slot()
    #map of optical flow, calculated at every pixel location:
    in_flow = Slot()
    #lists of optical flow vectors, same dimension as in_contours:
    out_flow = Slot()
    def propagate(self, cell, hint):
        SequencePropagator.propagate(self, cell, hint)
        #is the cell that notified us one of our inputs?
        if self.is_relevant(cell):
            #hint provides index and support of the input, to save rework
            seq_num,support = hint
            #do all required inputs have values at this seq_num?
            if self.input_ready(seq_num):
                #NB: Closest() is one way to compare support objects. it instructs
                #the cell to choose the value that has the most recent common
                #ancestor to support in terms of its derivation.
                areas,a_support = self.in_areas.select(seq_num,Closest(support))
                contours,c_support=self.in_contours.select(seq_num,Closest(support))
                flow,f_support = self.in_flow.select(seq_num,Closest(support))
                #create a new support object, noting how our output was derived and
                #the support objects it depends on
                support = Support('average-optical-flow-per-blob',
                                  depends=(a_support,c_support,f_support))
                #make sure our output hasn't already received an update with the same
                #sequence number and support. If it had, we could still submit an
                #update to it provided we did not change its value, but doing so would
                #waste effort.
                if not self.output_stale(seq_num, support):
                    #adkcv.averageOpticalFlow is in a fast library that does the actual
                    #work. I implemented it and many other external routines in C++,
                    #not shown
                    blob_flows = adkcv.averageOpticalFlow(flow,contours,areas)
                    #match shape of the pure python implementation
                    blob_flows = np.expand_dims(blob_flows, 2)
                    #finally, call the output cell's update method to register the update
                    self.out_flow.update(blob_flows,seq_num,support=support)
```

Listing 1: Implementation of a Propagator that computes average optical flow

Note that the propagator implementation in Listing 1 imposes directionality on the data flow of its connected cells. The cells connected to the slots `in_areas`, `in_flow`, and `in_contours` are treated like inputs, whereas the cell connected to `out_flow` is treated like an output. In general, the propagator framework imposes no such restriction on cells. Propagators are notified (resulting in their `propagate()` method being called) whenever the value of any of their connected cells changes. The `AverageOpticalFlow` example shown ignores updates from the cell designated `out_flow` because the inverse of its computation is undefined. We can learn about the `out_flow` cell from the `in_*` cells, but not vice versa, so the `out_flow` cell happens to be uninformative as an input to this propagator. Higher-level components presented in Section 4.3.3 exhibit bidirectional data flow.

4.3.2 Locating foreground regions

The goal of the first stage of processing is to identify regions of each video frame corresponding to objects of interest in the scene, such as pedestrians. The entire propagator subgraph leading from raw video frames to foreground regions annotated with optical flow maps is depicted in Figure 23, with example cell visualizations shown in Figure 24. In the processing involved in identifying foreground segments, the propagator architecture leads only to representational convenience and simplified program flow control rather effecting any paradigmatic change. In particular, as is the case with any framework that represents the graph structure of a computation while leaving control flow unspecified, the propagator architecture provides useful modularity, ease of parallelization, and convenient ways to visualize data flowing through the graph. At this first stage of processing, however, there are no computations that take advantage of the propagator architecture's bidirectionality, its potential for multiplicity of computational origin for cell updates, or iterative refinement or relaxation of cell values beyond a single update per cell per frame of video. In Section 4.4 I revisit ways to take advantage of such unique propagator capabilities in early processing.

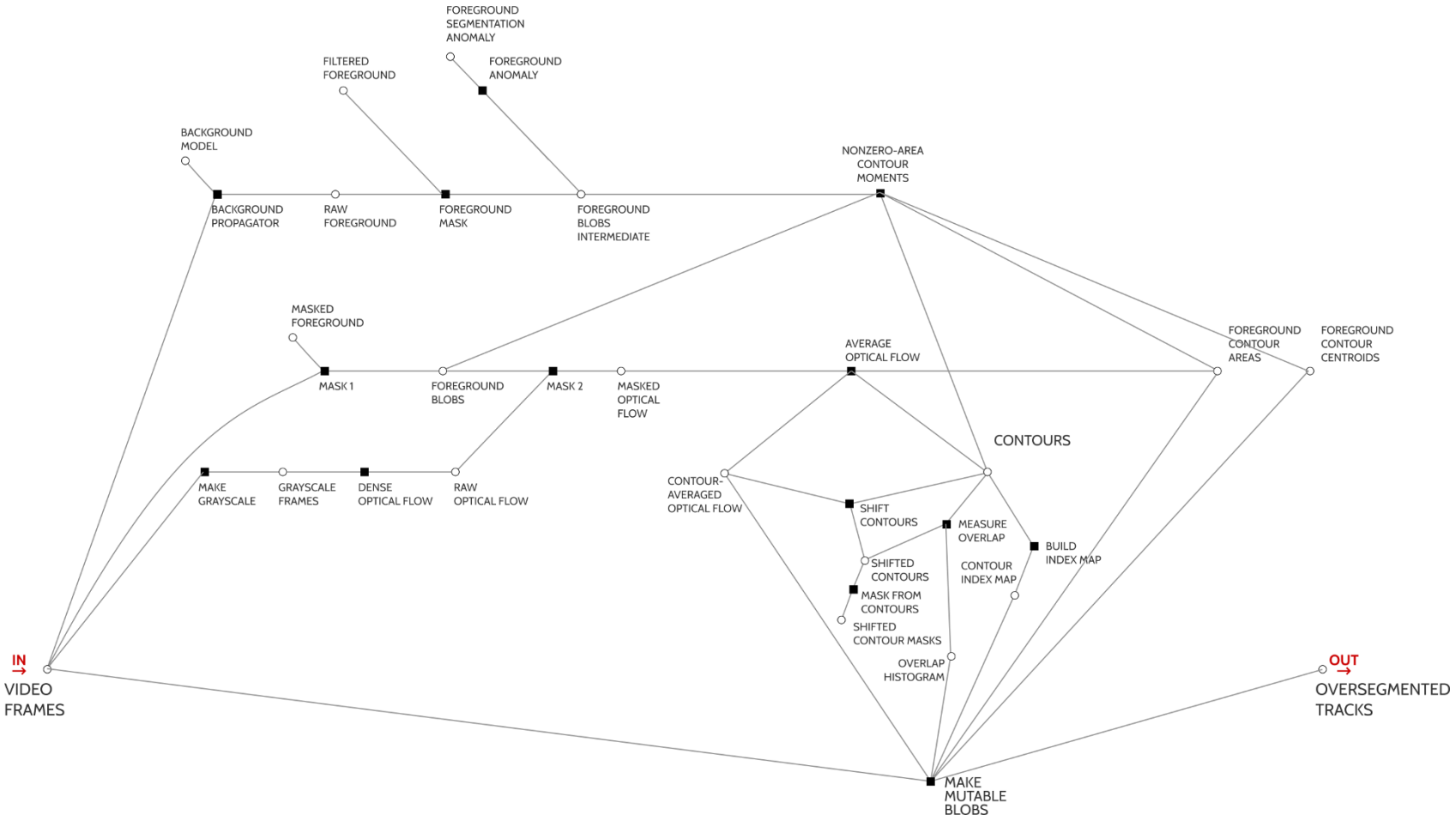


Figure 23: Flow graph of low-level processing

In this map of the first stage of visual processing, circles represent cells and squares represent propagators. Visualizations of selected cells are in Figure 24.

Having a stationary camera makes it possible to build a pixel-wise color model of the background over time, so that the foreground can be identified easily when pixel values do not fit the background color model. I used a method based on adaptive Gaussian mixture models by Zivkovic [2004] and Zivkovic and van der Heijden [2006]. The Zivkovic background subtractor extends the background subtractor of Stauffer and Grimson [1999] that models the background color as a mixture of K Gaussian distributions. The Zivkovic background model chooses the parameter K adaptively. The resulting pixel-wise foreground segmentation, shown in Figure 24, is noisy and requires smoothing. Smoothing is achieved via a combination of blurring, thresholding, and morphological operations implemented in the FOREGROUND MASK propagator as shown in Figure 23.

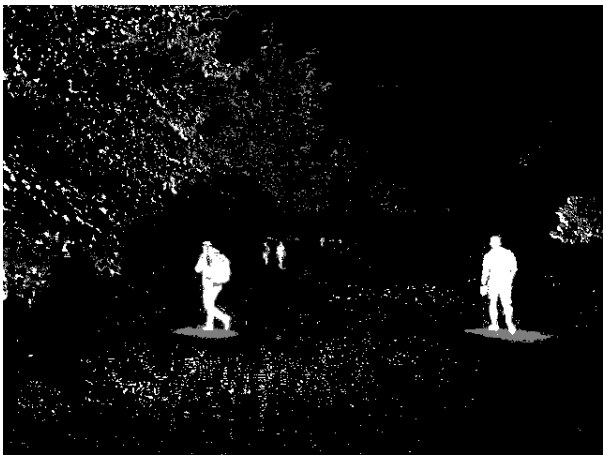
Another branch of the graph computes dense approximate optical flow via the method of Adarve and Mahony [2016] using the authors' GPU-based library. Although this method of computing optical flow is computationally cheap compared to other methods, it tends to produce spurious flow in regions nearby to moving regions, so it needs to be masked by clipping at the edge of foreground regions. The optical flow is used to compute average motion of each foreground region between consecutive frames, which in turn enables a primitive object tracker to generate short tracks of objects using conservative rules about intersection area versus union area in the forward-projected foreground regions from frame $N-1$ onto frame N. These oversegmented tracks are stored to disk for segmentation into longer tracks of objects by the next stage of processing. The short track segments are connected by weighted edges to form a directed acyclic graph (DAG), in which edge weights represent area of overlap between foreground regions on consecutive frames.

4.3.3 Tracking objects

Once the system has built up a DAG from overlapping foreground regions, the next question of interest is how to group the foreground regions into tracks, defined as time-ordered collections of foreground segments corresponding to the same physical object, or associated group of objects such as groups of pedestrians or people riding bicycles. Although foreground segmentation has simplified this problem, the problem remains complicated for several reasons:

- Foreground objects that occlude each other appear as a single foreground region.
- Inconsistent foreground segmentation can lead to splitting, merging, and momentary disappearance of foreground regions.
- Optical flow is a noisy, approximate proxy for actual motion of objects.
- Building appearance models to recognize specific objects is a deep subject.

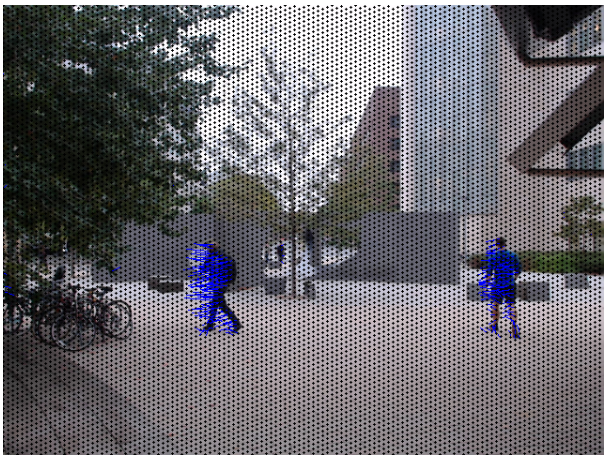
Many features that inform tracking of objects are expensive to compute, while others are computationally cheap. Furthermore, there are some decisions that may rule out any reasonable possibility that two regions correspond to the same object, for example, if the merger of those regions would require the corresponding objects to have moved unreasonably far between frames. Other decisions have the opposite effect: two foreground regions that remain identical between frames are unlikely to correspond to anything other than the same objects. The promise



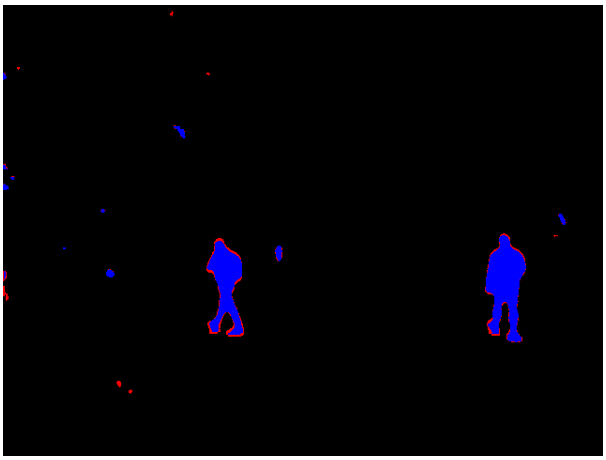
(a) Raw foreground

(b) Filtered foreground

(c) Masked foreground



(d) Masked optical flow



(e) Sifted contour masks

(f) Over-segmented tracks

Figure 24: Summary of low-level processing

The first stage of the system processes video frames with a background model to obtain raw (a) and filtered (b) foreground and segmented foreground regions (c). Using optical flow (d) to measure movement between frames, the segments are compared based on the degree of overlap between neighboring frames (e) in order to find an over-segmented representation of object tracks (f). Captions correspond to cell names in Figure 23.

of the propagator architecture is not only to make inference more accurate via exploitation of constraints, but also to prune unnecessary computation: if we are highly confident that regions should be merged together into a track, we should perform the most computationally frugal experiment to confirm our suspicion rather than going out of the way to quantify, in probabilistic terms, exactly how confident we should be.

In fact, it is possible to build propagator systems that have all of these desirable properties and several more. In this section I show how to build a decision process out of propagators and cells that builds up tracks by merging other tracks. The system can quickly rule out unlikely mergers without wasting computation figuring out exactly how unlikely those mergers are. The system can also quickly validate mergers that are practically certain to be correct, without wasting resources quantifying the probability of error. It can achieve these goals within a framework that leaves control flow up to a scheduler that can optionally decide to optimize performance using measured statistics about out-of-band aspects of propagators that do not bear directly on classification ability, such as their computational costs and resource requirements. The scheduler may make trade-offs between out-of-band factors and the propagators' expected influences on the final decision, because the order of execution of propagators is irrelevant to the final decision. Finally the system can optionally be tuned, with enough training data, to make statistically-justified decisions, in which the individual propagator components output probability values that reflect the measured statistics of some data distribution.

The intuition behind the decision system that evaluates mergers of tracks derives from cascade classifiers, which were first introduced as a part of the Viola-Jones face detector (Viola and Jones [2001]). In this section I briefly review cascade classifiers. I then present my work on taking cascade classifiers to the next level in the form of **symmetric cascades**. I describe how symmetric cascades generalize from the cascade classifiers used to detect faces in order to achieve the additional properties sought for merging tracks. Additionally, I describe how symmetric cascades can be applied in a machine-learning framework. Finally, I describe in detail how I use symmetric cascades to group over-segmented tracks into long, consistent tracks.

Review of cascades Conventional cascade classifiers apply a focus-of-attention approach to classification, in which a sequence of classifiers is applied to the parts of the image that have not been ruled out yet by previous classifiers. Each classifier in the cascade operates on a single feature. Each classifier is also tuned so that the false-negative rate is very low, even though the false-positive rate may be very high as a result of the tuning. The classifiers in the sequence are also ordered, so that computationally-cheap classifiers are executed first, and more expensive classifiers are reserved for sub-regions of the image that are not quickly ruled out by the computationally-cheap classifiers. A variant of AdaBoost (Freund and Schapire [1997]) is used to select the sequence of classifiers from a much larger set of candidates, so that the resulting boosted classifier is very unlikely to produce a false positive.

A useful intuition for thinking about cascades is to consider each individual classifier in the cascade as a strong rejector and a weak acceptor. It can decide to **reject** or **defer**, and a sufficiently long sequence of **defer** decisions for the same region by different classifiers constitutes an accept with high probability. In more specific terms, suppose the false-positive rate, defined in this context as the proportion of the time the object in question is absent and classifier i decides to **defer**, is e_i. Assume that the decision of each classifier is approximately independent of

the others. That assumption is reasonable if the feature representation concisely represents the independent factors of variation in the target class. Then the overall false-positive rate, E, of the combined classifier is given by:

$$E = \prod_{i=1}^{K} e_i \tag{2}$$

Even if the individual e_i are high, E becomes geometrically lower with each classifier added to the cascade. Improving matters further, the downstream cascade stages can be more computationally expensive without adding much cost to the whole cascade, because they are run with much lower frequency than the earlier stages. They can therefore afford to have lower false-positive rates than the earlier stages, and dramatically improve precision at low cost.

Symmetric cascades In order to empower propagators to track objects, I take cascades to the next level by incorporating two changes. I call a cascade incorporating the changes a **symmetric cascade**. The changes enable efficient object tracking but I expect that symmetric cascades will be useful generalizations of cascades for other problems as well. The changes are as follows:

1. In addition to being able to **reject** or **defer**, units in a symmetric cascade may **accept**, halting computation on a sub-problem with a positive result.

2. In the case of **defer**, units in symmetric cascades update a numerical measure of confidence in the outcome.

The first generalization permitting **accept** allows more efficient pruning of the computation by permitting computation to stop when one unit performs a test that results in practical certainty of a positive outcome. The ability to accept early imparts a new symmetry to the cascade's computation: it can stop early and save work by either accepting or rejecting, whereas an ordinary cascade is asymmetric in that it can only stop early if it rejects. The second generalization requiring numerical confidence updates rather than discrete answers allows several out-of-band optimizations to take place: computational complexity can be traded for accuracy if need be, and reliable bounds can be placed on the final confidence without doing the work to determine an exact numerical value of that confidence.

To make things concrete, let T be a binary random variable. Let $p_T(i) \equiv P(T = 1|F_0 = f_0, ..., F_i = f_i)$, the probability that $T = 1$ given observations of some set of features up to and including feature measurement f_i. The role of the symmetric cascade is to predict the value of T by estimating this probability, while considering as few features as possible to achieve a desired confidence in the outcome. That is, the symmetric cascade computes $\hat{p}_T(i)$ for $0 \leq i \leq N$ such that $\hat{p}_T(i)$ is a good estimate of $p_T(i)$, and furthermore that $p_T(i)$ is an estimate of $p_T(N)$ that satisfies some configurable bound. An instance of a symmetric cascade consists of a set and ordering of features F_i to evaluate, and rules for determining when to stop the computation when the outcome of T can be predicted with high enough confidence.

I first discuss how I implement a symmetric cascade. A useful aid in implementing symmetric cascades within the propagator framework is to separate the required state variable apart into

scalar values that change monotonically during a computation. By definition, cells record *monotonically increasing information about values* rather than values themselves. While it is not strictly necessary for cells to contain monotonically-changing scalar values, it does simplify bookkeeping and facilitate code reuse, and, as I show later in this section, has additional benefits for the interpretation of symmetric cascades from a machine-learning perspective. Recall that cells record information about the parameters of a probability distribution $\hat{p}_T(i)$ that approximates the true data distribution $p_T(i)$. I represent the approximate distribution $\hat{p}_T(i)$ in terms of two additional parameters, the accept accumulator A_i, and the reject accumulator R_i:

$$\hat{p}_T(i) \equiv \frac{A_i}{A_i + R_i} \tag{3}$$

A_i and R_i are defined to have identical domain $[0, 1]$. Each pair of these variables, except for the error condition in which they are both 0, specifies a value of $\hat{p}_T(i)$, and every $\hat{p}_T(i)$ determines A_i and R_i up to a common scale factor. A_i and R_i are computed as the respective products of factors supplied by each each unit in the symmetric cascade:

$$A_i \equiv \prod_{k=1}^{i} a_k, \quad R_i \equiv \prod_{k=1}^{i} r_k \tag{4}$$

Each unit in the symmetric cascade emits values for a_i, $\$r_i,\$$ or both; the requirements of the cascade unit's output are that the unit must not emit $a_i = 0$ and $r_i = 0$, and the domain of both a_i and r_i is $[0, 1]$. There is redundancy in these variables insofar as they represent $\hat{p}_T(i)$. For example it is possible to configure each unit so that it always outputs 1 for either a_i or r_i without loss of generality. The cell responsible for representing $\hat{p}_T(i)$ at every stage in the cascade need only store the current values of A_i and R_i. Because the updates are all in the range $[0, 1]$, their products A_i and R_i are monotonically decreasing in value, and so the valid interval of each of these variables is represented succinctly by the variable's current value. The final essential technical detail needed to make the system work is that it must be known a priori whether each propagator in the symmetric cascade is capable of emitting values of a_i and r_i such that $a_i < r_i$, or such that $a_i > r_i$, or values that obey neither restriction. In practice this is easier than it may seem: certain features may weigh strongly for or against the expected outcome of T if present, but provide no information if absent.

I now show how these definitions lead to the sought after properties in the generalized cascade. Fast acceptance and rejection are achieved by having a unit emit 0 for one of a_i or r_i, which collapses the value of $\hat{p}_T(N)$ to a degenerate probability distribution. Emitting $a_i = 0$ causes immediate rejection, whereas emitting $r_i = 0$ causes immediate acceptance. All processing on a value may stop after receiving such an update, because no subsequent valid update can have an effect on the value of $\hat{p}_T(N)$.

Because A_i and R_i are products of updates, the order of the updates has no effect on their values. This gives the scheduler leeway to structure the computation in serial or in parallel, and to order the individual propagator execution however is most convenient. Formally, in terms

of the definitions presented here, this means that any feature set of length N potentially gives rise to $N!$ distinct symmetric cascades that each estimate the function $p_T(N)$. Many scheduling strategies can emerge to choose the order of the feature propagators. A scheduler could, for example, execute first the propagators that are likely to collapse the distribution, increasing the chance that it may stop work early. A related strategy is to order the executions so that propagators that are likely to have large influence on the value of $\hat{p}_T(N)$ are executed with high priority. Computation can optionally be halted early if the value of $\hat{p}_T(i)$ falls below a rejection threshold or rises above an acceptance threshold. Of course, unlike in conventional cascades, there is no general guarantee in a symmetric cascade that $\hat{p}_T(j > i)$ will remain below or above such thresholds as computation continues. Thus it must be understood that a strategy of stopping when a threshold is crossed merely approximates the process of running all propagators to completion, and constitutes a trade-off between correctness and efficiency. It is nevertheless a useful tool for a propagator system to have in its arsenal.

If no sacrifice of correctness for efficiency is acceptable, there is a more conservative strategy that still offers efficiency benefits over a conventional cascade. Suppose that during execution a scheduling process observes that $\hat{p}_T(i)$ rises above an acceptance threshold L_A. Because it is known a priori which propagators have the potential to emit values that may cause the running estimate $\hat{p}_T(j > i)$ to only increase, to only decrease, or to either increase or decrease, the scheduler can skip propagators that can only cause an increase. That is, if all propagators run that can emit a_j, r_j pairs potentially causing $\hat{p}_T(j > i) < \hat{p}_T(i)$, and the working estimate $\hat{p}_T(j)$ remains greater than threshold L_A, it is safe to halt the computation: no amount of further work can cause the estimate of $\hat{p}_T(j)$ to fall below L_A. An analogous strategy exists for rejection thresholds.

If the scheduler has access to probabilistic bounds on the values of a_i and r_i emitted by each propagator, it can use such statistics to implement a compromise between the simple, stop-at-threshold approach and the conservative, rule-out-alternatives approach. For example, suppose that the working estimate of $\hat{p}_T(i)$ rises above acceptance threshold L_A, and the probabilistic bounds on the values of $a_{j>i}$ that can be emitted by the remaining propagators indicate that with probability q, it will remain true at the final step N that $\hat{p}_T(N) > L_A$. Then, we may terminate if q exceeds a meta-certainty threshold. In this case q quantifies certainty based on ongoing self-measurement about whether the system will remain confident about its answer if it were to execute all propagators that could potentially reduce its confidence. If q passes muster we can halt execution of propagators and be reasonably confident in the answer at step i.

It has always been a difficult task to design classifiers that can be optimized on the fly for different measures of performance, such as speed, resource use (especially out-of-band resources, e.g., battery energy), confidence in answers, and flexibility to changing operating conditions. It is much easier, generally speaking, to measure aspects of performance, and nearly all attributes of propagators in the generalized cascade that bear on performance can be measured empirically. When computations are broken up into reorderable sub-units as they are in the symmetric cascade, it becomes possible to use data from such measurements to restructure classification pipelines on the fly to optimize for whatever performance metrics are most relevant under a set of operating constraints. A scheduler process implementing this kind of flexibility could start out by running all available propagators in some arbitrarily chosen order, and measure the statistics in question over many executions. Then, once it collects sufficient information, the scheduler can begin planning the order of execution to optimize a performance measure. It could, for example, order propagators by increasing resource use, by increasing expected value of a_i and r_i, or some

combination thereof.

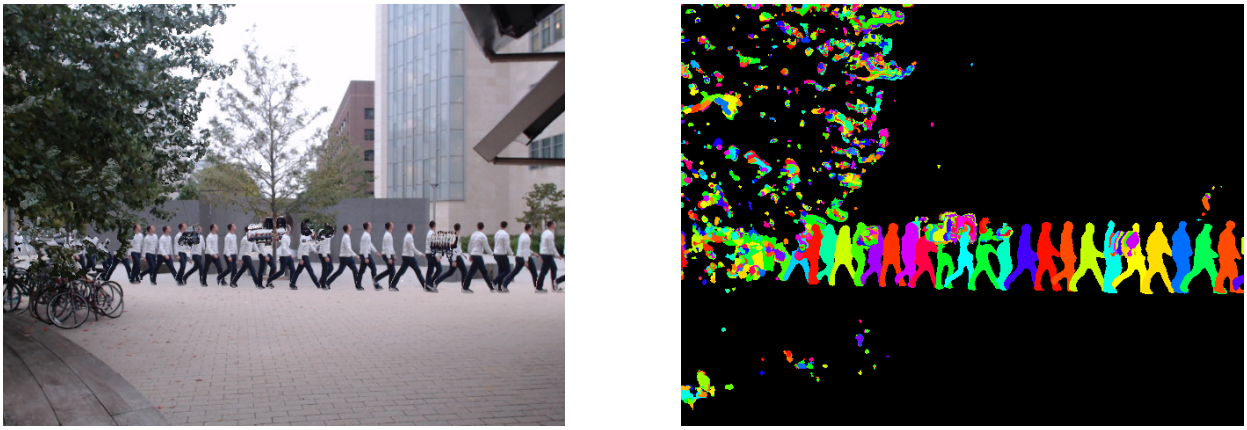
Throughout the discussion of generalized cascades, I have taken for granted that $\hat{p}_T(N)$, the approximation of the true data distribution described by $p_T(N)$, is a good approximation. The final and perhaps most profound aspect of symmetric cascades that I discuss here is that symmetric cascades can be learned from training data, in a way that validates this assumption.

In a conventional cascade, it is not necessary to know the conditional probability of a positive outcome at any given step in the cascade. If the outcome is negative (**reject**) it is assumed that the probability in question is negligibly small, and if the outcome is positive (N **defer** answers in a row) then, although we can approximate the conditional probability of error as the product of the false-positive rates of all detectors, this estimate provides little actionable information as it should always be the same. By contrast, a symmetric cascade is expected to stop early in both **accept** and **reject** scenarios, and should it execute all N stages without encountering a stopping condition, that would suggest that the question of interest is difficult to answer. We would not expect the conditional probability at stage N to be close to 0 or 1, and it would be helpful if it in fact reflected measured statistics about some training data, so that it could be used as input to a different classification strategy that can make use of the uncertainty measure. Another reason to want the conditional probability modified by each stage in the symmetric cascade to be grounded in measured statistics is that acceptance or rejection may be due to a threshold computation, as discussed, rather than due to the distribution collapsing to exactly 1 or 0, and in such cases it is desirable to have confidence thresholds that are interpretable as tolerated error rates of the cascade.

When considering how to accurately represent $p_T(i)$ via the symmetric cascade, the benefit of the definition $\hat{p}_T(i) \equiv A_i/(A_i + R_i)$ becomes clear. This is the form of a naive Bayes classifier. Therefore, assume the naive Bayes criterion that all $P(F_i|T)$ are independent. Let $A_0 \equiv P(T = 1)$, and $R_0 \equiv P(T = 0) = 1 - P(T = 1)$. These priors can be measured empirically from training data. Let a_i and r_i respectively be approximations of $P(F_i = f_i|T = 1)$ and $P(F_i = f_i|T = 0)$, resulting from fitting parameterized probability distributions to the features observed in some training data. With these definitions and under the naive Bayes assumption, the symmetric cascade is an instance of a naive Bayes classifier, where the features are evaluated separately and according to the operational needs enforced by the scheduler. Additionally, features are preferentially selected to cause the cascade to stop processing by collapsing the probability distribution to a degenerate probability distribution. These features are more naturally thought of as weak classifiers, similar those that make up ordinary cascade classifiers except that they have as many as three outputs. The classifiers have the property that they may either have a near-zero false-**reject** rate and a high **defer** rate, *or* a near-zero false-**accept** rate with a high **defer** rate. They may also have neither or both of these attributes. In any case, the probabilistic interpretation of **defer** must be well characterized, unlike in the ordinary cascade where it need not be known.

In this section I have introduced **symmetric cascades**, a generalization of cascade classifiers that can stop processing on both **accept** and **reject** outcomes. I described how to implement symmetric cascades within the propagator framework, and how their use can lead to new efficiency and adaptability to varying operating constraints. I explained how symmetric cascades can be couched as instances of naive Bayes classifiers, and can therefore be learned from training data. In the next section I describe a practical application of symmetric cascades in object tracking.

Applying symmetric cascades to object tracking The specific object-tracking problem of interest is illustrated in Figure 25. Foreground labeling produces a foreground mask for each frame in a video. Each set of connected components of the foreground mask, called a blob from here on, corresponds to one or more objects in the observed scene. The goal is to link together blobs that correspond to the same object or set of objects into tracks, and to discard blobs corresponding to misclassified background, or segmentation anomalies.



(a) Foreground composite

(b) Foreground blobs

Figure 25: Raw input to the tracker
The time series visualized at 10-frame intervals in (a) is shown cast in its lowest-level representation in (b). Each foreground segment, or *blob*, is initially assigned a random unique identifier, represented by the color of (b). The tracking program must link the blobs representing the same object together, across the space and time coordinates.

This framing of the tracking problem is a simplified version of the tracking that humans easily perform. We do not typically become confused about object identity when objects partially occlude each other or move in close proximity to one another, but the system is not required to make sense of such scenarios beyond distinguishing tracks of the individual objects from tracks of the segments formed by groups of partially-occluding or nearby objects.

The system builds up consistent tracks of objects by proposing candidate pairs of over-segmented tracks that might correspond to the same object, and then vetting those pairs using a symmetric cascade. Figure 26 depicts the merging process for a track of a pedestrian walking towards a partial occlusion boundary. Each binary merge in the tree corresponds to the decision made by a symmetric cascade element. The decisions are recorded in the track's `Support` data structure, from which Figure 26 was generated.

Proposals are generated when a pair of tracks is nearby in space-time volume. The proposal procedure considers the longest in the current list of tracks first (called the seed track). For each seed track, the process builds a list of tracks that occur during overlapping time segments. Only overlapping tracks that have all of their blob centroids within a blob-distance threshold of the centroids of the seed track are considered. To account for small time discrepancies caused by short-duration occlusions and foreground-segmentation anomalies, the system interpolates centroids and areas over a window of 5 frames. The blob distance d_B between two real or interpolated blobs a and b with centroids $\boldsymbol{c}_a$ and $\boldsymbol{c}_b$, respectively, and areas A_a and A_b, respectively, is

as follows:

$$d_B(a, b) \equiv \frac{|\boldsymbol{c}_a - \boldsymbol{c}_b|}{\sqrt{A_a + A_b}} \tag{5}$$

That is, the distance between the centroids is normalized by area. Note that this is not a metric because it does not satisfy the triangle inequality. It is nevertheless a useful way to measure meaningful proximity between blobs: in practice, large or near-field objects produce more distant isolated patches of foreground than do small or distant objects. This blob distance is a reasonable first-pass cost function for eliminating pairs of blobs that should not be considered for merger, allowing the number of merge proposals to be closer to linear rather than quadratic in the number of tracks. Because the number of tracks scales according to the total length of the video, linearity in the number of tracks is necessary for real-time performance. This implementation does not run in real time, but because real-time performance is an essential aspect of vision systems, there must be hope of making it linear.

The propagators in the symmetric cascade then use several features to decide whether tracks proposed for merger should actually merge. All are engineered, heuristically-motivated features, and augmenting this strategy with feature learning is an area for future development. The features fall into three categories: geometric features, motion features, and appearance features.

When computing motion features, the cascade must take into account that the optical flow measurements are noisy and unreliable. The only motion feature used is a time-averaged euclidean distance between average optical flow per blob. For every pair of (actual or interpolated) blobs in the tracks considered for merge, a cascade element computes the euclidean distance between the vector resulting from integrating optical flow over all of the area contained in each blob. The time averaging uses a window of fixed length (30 frames, which amount to 1 second of video). The result is a weak measure of similarity that is fairly robust to noise, taking into account direction and magnitude of the motion of blobs.

The only appearance feature used in the implementation is a color histogram in HSL color space, with 32 histogram bins for hue and 16 bins each for saturation and lightness. All foreground area in each track contributes equally to the histograms. To compare two histograms, the program normalizes each histogram and then computes the sum over the pair's bin-wise minimum.

The geometric features used to vet track mergers are the total number of connected foreground components in each blob, N_c, the ratio between the minimum and maximum blob distance between the two tracks, R_b, and the maximum contour separation in blob distance, D_M. Penalizing high values of N_c prevents a snowball effect arising from merging too many spatially distant blobs from dominating the merge process. R_b compares the maximum and minimum values of d_B over the frame range of the tracks. Too high a value suggests that the blobs are moving independently of one another and should not be merged. To calculate D_M, a variant of the blob distance is used that measures from edge points of blob contours rather than blob centroids. D_M is the largest (area-normalized) minimum distance between boundaries of any two blobs in the proposed merged track.

$$D_M \equiv \max_{B_1,B_2} \left(\min_{C_1 \in B_1, C_2 \in B_2} d_m(C_1, C_2, B_1, B_2) \right) \tag{6}$$

Here, C_1 and C_2 are contours: sets of points that define the boundaries of the connected components comprising blobs B_1 and B_2, respectively. D_M is defined to be the maximum value over the proposed merged track of the minimum d_m value of any one of the pairs of blobs considered in the merger. The function d_m, in turn, is an area-normalized distance but instead of being computed between centroids of blobs it is computed between the closest points on contours:

$$d_m(C_1, C_2, B_1, B_2) \equiv \frac{\min_{\boldsymbol{p}_1 \in C_1, \boldsymbol{p}_2 \in C_2} |\boldsymbol{p}_1 - \boldsymbol{p}_2|}{\sqrt{A_{B_1} + A_{B_2}}} \tag{7}$$

The symbols A_{B_1} and A_{B_2} represent the areas of two blobs considered for merger. Overall, the feature D_M is the maximum, over each pair of blobs B_1, B_2 in the proposed merged track, of the minimum area-normalized distance between a point on some contour $C_1 \in B_1$, and a point on some other contour $C_2 \in B_2$. Preventing mergers of tracks with too large D_M prevents merging blobs whose centroids appear close, but whose individual foreground contours are all distant.

The final procedural detail required to produce bounding boxes for the tracked objects is that the bounding boxes must be smoothed in order to exhibit enough regularity to be used for reliable geometric measurements. An order-2 Butterworth filter with a cutoff frequency of 0.05 times the Nyquist rate (equivalent to 0.75 Hz) provides adequate smoothing when applied independently to the x and w bounding-rectangle parameters. The y and h parameters use a filter with a lower frequency cutoff of 0.17 Hz. The filters are applied forward and backward to each track, ensuring no phase shift. This non-causal filtering is one of many smoothing strategies that could be used, so it does not present an insurmountable obstacle to real-time performance.

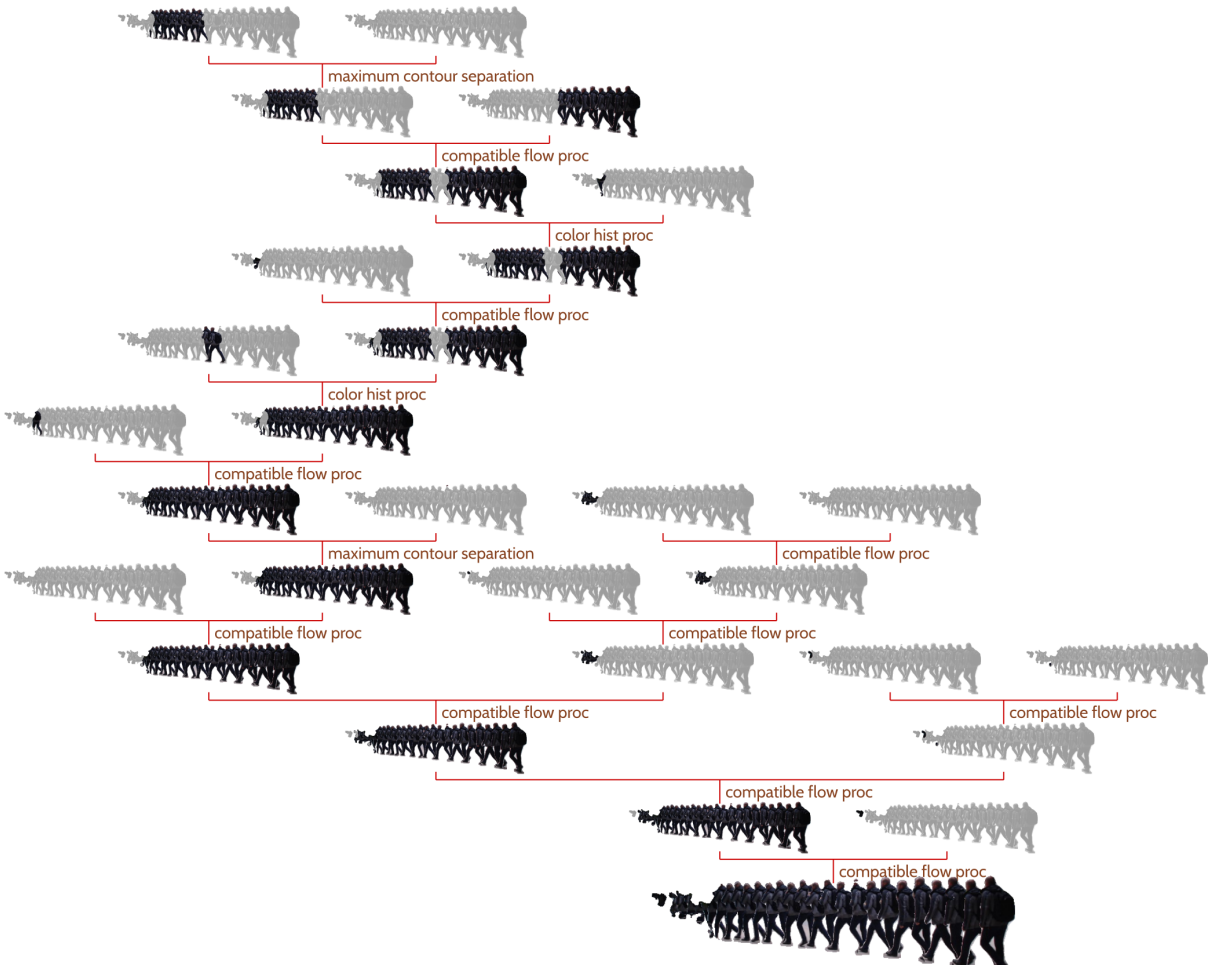


Figure 26: Output of track merging via symmetric cascade
Each merge decision in the binary tree is attributed to the stage of the symmetric cascade that caused the merge. All decisions are derived from the track's support data structure. Dark patches represent foreground objects that have been merged at that point in the tree, and light patches represent all other non-merged tracks. The tree represents a complete merge process, from short over-segmented tracks produced by simple rules to the final track. All leaves represent unique tracks, even though some of the differences have too small area to be visible in the format used for display.

Locating occlusion boundaries By noting the blobs that form endpoints of tracks, it is possible to accumulate information about where objects appear and disappear. Such information, which has semantic interpretation as *source* or *sink* loci of tracks, initiates the process of learning about background scene geometry from object tracks, and then propagating that knowledge back to the foreground. This process begins to expose the power and expressiveness of propagator representations for perception beyond purely mechanical benefits such as modularity and ease of parallelization. The existence of source and sink loci can be determined from evidence in the form of tracks, but it can also be used to interpret the tracks themselves. Where should this mutually-reinforcing process begin, and how can we ensure that it is not based on circular reasoning? How can external information, of the type that story-understanding systems or additional modes of perception can provide, integrate seamlessly into this process? I address these

questions in this section.

Cells maintain information about source and sink loci and occlusion boundaries. The cell of class `SourceSinkLoci` holds a set of bounding rectangles formed by aggregating the first and last frames in tracks. `SourceSinkLoci` cells accumulate these bounding boxes.

Additional cells build up a lower-level type of occlusion information. Whereas `SourceSinkLoci` represent places where moving objects are likely to appear and disappear, they do not contain fine-grained spatial locations of occlusion boundaries, and they cannot represent small occluders such as the tree trunk in Figure 25. Cells of class `HistMap` hold, for every pixel location in an image, one of the following types of histograms:

- HSL color histograms of all blobs that have had an edge (boundary pixel) at the given location, accumulated over all time, with 32 hue bins and 16 bins each for lightness and saturation.
- Optical flow direction histograms, containing 8 direction bins. These histograms accumulate the averaged optical flow vectors for blobs. The histograms corresponding to the pixels at the edge of blob contours are updated with the averaged optical flow of the blobs.
- Edge-present frame-count histograms, in which the bin accumulators at each pixel location are updated whenever the corresponding pixel remains on the edge of a blob for a certain number of frames. The 6 bins represent approximately exponentially increasing frame counts: 4-7 frames, 8-15 frames, 16-31 frames, 32-63 frames, 64-127 frames, and a catch-all bin for 128+ frames.

Figure 27 depicts the location-mapped histograms of a `HistMap` cell that contains edge-present frame-count histograms. Figure 28 depicts optical flow and HSL histograms, which are also stored in `HistMap` cells at every pixel location. Additionally there are `ListMap` cells, which map arbitrary lists to pixel locations. The two types put to use are:

- Lists of HSL histograms, for each blob that has ever had an edge at the given point. Unlike the histograms in the histogram cells, these histogram accumulators are never again modified after they are stored.
- Lists of all distinct frame sequence numbers that a given pixel has been included in a blob edge.

All together, the three types of histogram cell and the two types of list cell provide all the information needed to derive a pixel-level map of occluders. To build intuition for why this works, suppose there is a region of the image that often experiences poor foreground segmentation, such as a region containing movable vegetation. The edge duration histograms may accumulate large values here, but the overall color histogram and list of color histograms will all contain similar values corresponding to the colors of the moving vegetation. At edges of occluders of interest (places where pedestrians appear and disappear in the scene), there is more variation in the color histograms, and more concentrated peaks in the duration histograms. Loci of frequent occlusions of interest will have sharp spatial peaks as depicted in Figure 27.

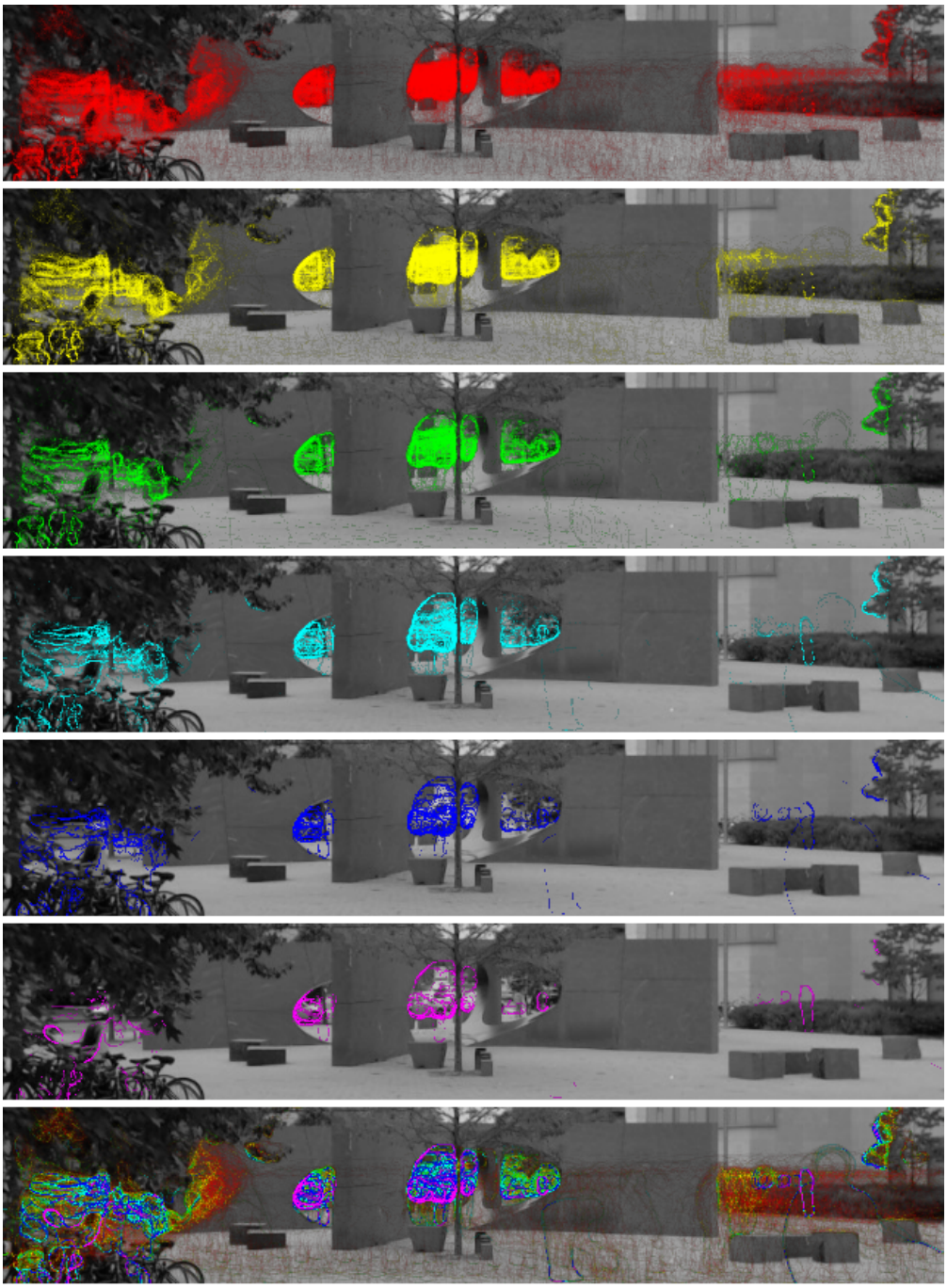
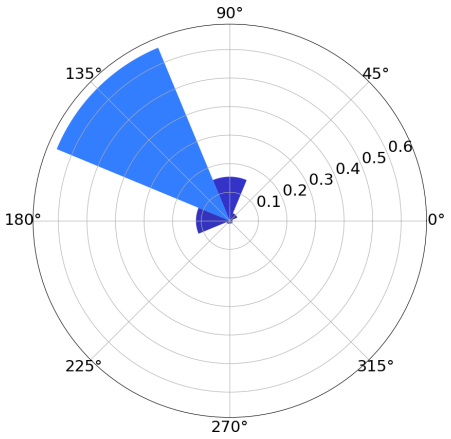


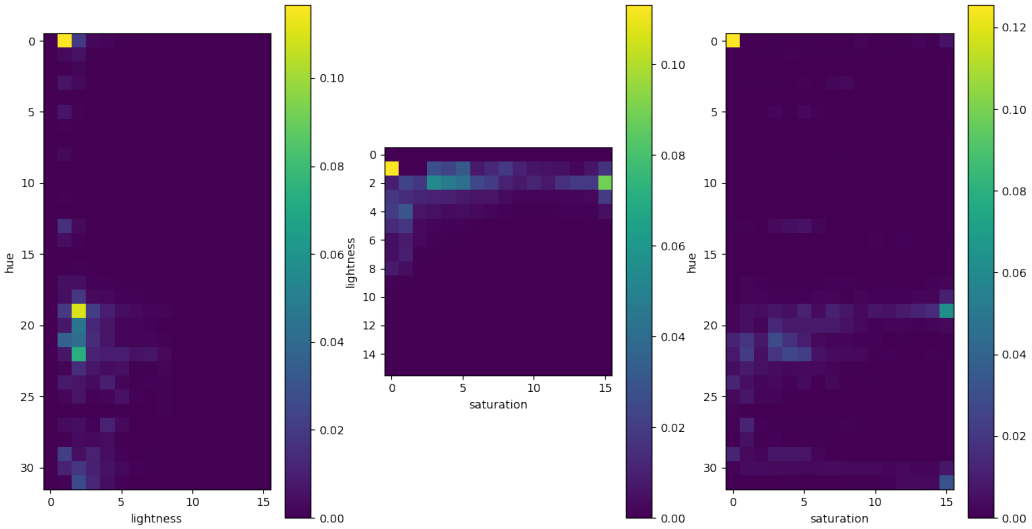
Figure 27: Edge-present frame-count histogram composite

Colors represent histogram bin and color intensities represent histogram value in this composite of an edge-present frame-count histogram generated from approximately 20 minutes of processed video. In this representation, which is used by `HistMap` cells, histogram bins are incremented whenever the corresponding pixel location contains an edge for each of the following ranges of consecutive frames (from top to bottom) 4-7, 8-15, 16-31, 32-63, 64-127, 128-∞. The strip at the bottom is a layered composite of the bin images.

(a) Foreground region



(b) Optical flow orientation histogram



(c) HSL histogram cross-sections

Figure 28: Optical flow and color histograms
The representative example foreground blob (a) produces the optical flow histogram (b) with 8 direction bins and the color histogram shown in three projections (c). The HSL histogram has 32 equally-spaced hue quanta and 16 quanta each for lightness and saturation (for a total of 8192 bins). Each histogram is stored at every pixel location in a dense HistMap cell, analogous to the way edge-present frame-count histograms are stored (as depicted in Figure 27).

Propagating geometric information The vertical orientation of the camera in this setup makes it easy to estimate the distance from focal plane, Z, given the height in pixels and the height in meters of an object known to be standing vertically. Furthermore in many environments such as the one used in this example, it is reasonable to approximate the ground as planar

or piecewise planar. Applying least squares optimization assuming that every pedestrian is of average adult height finds the approximate ground plane, as shown in Figure 29.

(a) Pedestrians and ground plane

(b) Ground plane error

Figure 29: Output of least squares estimation of the ground plane
Seventy pedestrian tracks of at least 150 frames each provide the image heights and bounding boxes used to estimate the ground plane, shown in green in (a) with overlaid track samples from many tracks. Grid units are nominally 1 meter, and the grid is shown only where supported by measurements. Pedestrian heights are assumed to be 1.68m, leading to the situation (b) in which people taller than average appear to float above the estimated ground plane.

The program can do better by jointly optimizing the parameters of the ground plane and the pedestrian heights. This optimization makes use of the observation that a person's measured height doesn't change by very much as the person walks, in order to link together the real-world heights of the many bounding boxes measured from the same pedestrian track. In this simple example scenario, it is straightforward to apply global optimization to a cost function that penalizes deviation in height from the national average and distance of the ground-support points from the estimated ground plane. This solution would work in the case of estimating pedestrian heights and ground-plane parameters, but it does not naturally scale to large complex problems with more unusual types of constraints and many components and variables.

Instead of explicit global optimization, the propagator methodology seeks a type of implicit local optimization that arises naturally from the connections and constraints in the network. One way to do this in the example is to have separate cost functions for the ground plane parameters and for the individual pedestrian heights. Then the propagators perform a relaxation algorithm: alternating between estimating the ground-plane parameters holding the pedestrian heights fixed, and the pedestrian heights holding the ground-plane parameters fixed. Figure 30 depicts the results of that relaxation.

(a) Pedestrians and relaxed ground plane (b) Ground plane error resolved

Figure 30: Output of propagator relaxation of the ground plane
After applying a relaxation propagator network, variables in the system reflect the constraint that pedestrians' feet rest on the ground. The relaxation improves the ground-plane solution, and it better estimates the height of each individual pedestrian.

In order to implement relaxation while preserving a type of monotonicity, the implementation relies on a type of cell that holds a collection of values. Cells store every update that does not have identical support as one of its members, and ideally never discard any values. A value in the example case refers to an instance of ground-plane parameters or an instance of pedestrian height assignments. An update with a different value but the same support as an existing member of the cell is an error condition. A short-circuit mechanism prevents propagators from redoing work that has already been done, because the generated support tree for the value that would result from that work would already be present in the target cell. In the case of the relaxation algorithm, propagation stops when convergence occurs.

To take things a step further, propagation can incorporate the locations of source/sink loci of tracks. The network infers locations of occluders from accumulated histograms of foreground regions like those depicted in Figures 27 and 28 as well as from statistics of bounding boxes at the beginnings and ends of tracks in time. The results are shown in Figure 31. When a pedestrian appears from behind an occluder of known Z position in the scene, the occluder sets a lower bound on the appearing person's height. Observations of foreground objects both occluding and becoming occluded by a background object of known Z position can bound the foreground object height both from above and from below.

Figure 31: Locations of occluders
Intensity of blue shading corresponds to number of tracks observed to appear or disappear near the indicated region. The red region marks the occlusion boundary having the the highest confidence from among those found in the scene. The Z coordinate of the base of the occluding object is manually annotated with ground truth measured using stereo disparity.

One way to use this type of occlusion constraint is to build up evidence for the distance of the occluder and apply relaxation as in the ground-plane example. Another way is to use known measurements of the occluder's distance to make inferences about the heights of pedestrians that interact with the occluder. In this case, setting the occluder's height has rippling effects: the pedestrians that are seen to interact with the occluder have bounds placed on their heights. The new bound assignments cause automatic restarting the ground-plane relaxation process and solving for new plane coefficients and new heights for the other pedestrians. A propagator network implementing that process is illustrated in Figure 32.

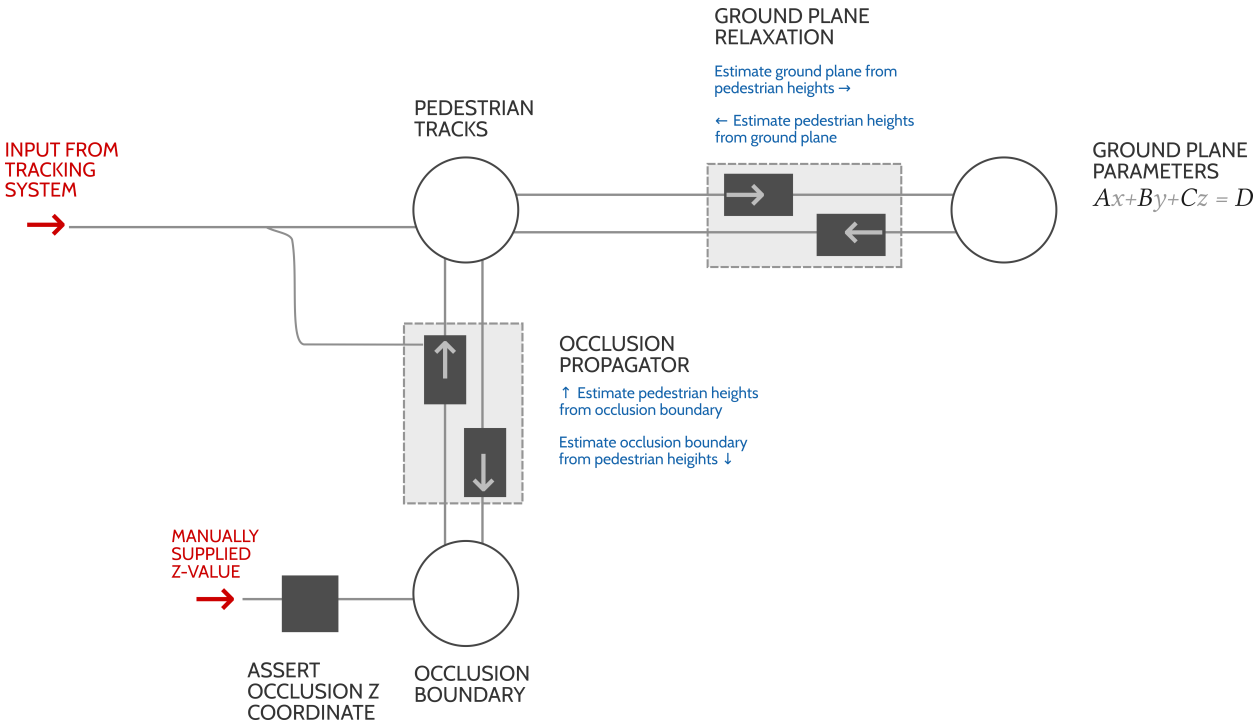


Figure 32: Top-down propagator network

The portion of the propagator network that can interface with a high-level system accepts Z coordinate values for objects' occlusion boundaries that other parts of the propagator system have detected. Setting the Z coordinate of the obstacle affects the values of other cells in the system by causing a relaxation algorithm to update pedestrian heights and ground-plane parameters.

Applying this network by assigning the value of 24.4 meters to the Z coordinate of the occluder shown in red in Figure 31 has little effect on the ground plane solution, using the small sample of 70 successfully tracked pedestrians from the approximately 20 minutes of processed video. Out of the 23 tracks that start or end at the occluding wall, none undergo a change in height due to the updated lower bound; their heights had already been relaxed to values above that imposed by the constraint. Of the 36 tracks that pass in front of the occluding wall, exactly one undergoes a decrease in height due to the updated upper bound, from 1.64 meters to 1.60 meters. Samples from the affected track are shown in Figure 33. From this image it is clear that the track should not have been constrained by the geometry update; its small appearance in the vicinity of the occluder is due to background segmentation failure. The good news in this case is that the change in height has negligible effect on the ground plane parameters.

Figure 33: Track affected by propagation
The only track from a sample of 70 tracks that is affected by propagation of the labeled occlusion boundary exhibits segmentation errors. All other tracks are successfully determined to be unaffected by the constraint imposed by the occlusion-boundary distance information.

This process has implications for connecting story understanding systems to perception systems. Suppose that a story understanding system has the knowledge *the Collier Memorial statue stands 3.0 meters tall at its eastern edge*, and that some vision component, not modeled here, can recognize the eastern edge of the statue. Then the propagator system can infer the Z coordinate of the edge of the statue, propagate the knowledge to pedestrians occluded by or occluding that edge and then to the ground plane, which has potential to influence the way other scene objects are interpreted. The implications extend to hypothetical reasoning as well: the story understanding system can issue a hypothetical statement to the propagator network and measure its propagated effects against known quantities. Because saving and restoring the entire state of the perceptual network in order to test a hypothesis is impractical, the propagator architecture can relatively efficiently prune out all inferences that depend on the hypothetical statement by inspecting supports, and, due to the monotonicity property, guarantee that no other values could have been polluted by the hypothesis testing process.

4.4 Discussion

The initial steps toward building a propagator-based vision system and the qualitative results obtained from that system show promise, and invite further development. The initial implementation I described here demonstrates how top-down influence, in the form of assertions about object dimensions and locations, can influence the perceptual processing of a scene and result in deductions about the dimensions and locations of other objects in the scene, based on the way that objects interact with each other. The implementation realizes one of the ways that a high-level cognitive process can reflect its knowledge back onto its perceptual apparatus to learn something new: by forming a hypothesis, and determining if the propagated effects of that hypothesis corroborate or contradict other knowledge that the high-level process has. This represents significant evidence in favor of the alignment hypothesis motivated by observations about natural vision systems in Chapter 1. In the implementation I described here, externally supplied measurements, which serve as a stand-in for expectations about the world arising from story-level processing, reflect back onto an intermediate-level representation in which tracked objects are represented by features like contours, color histograms and bounding boxes, and as a result of this reflection, the system is able to refine both the high- and intermediate-level descriptions and provide justification for the refinements. The alignment hypothesis also calls for *pervasive align-*

ment, that is, it is predicated on alignment being ubiquitous at all levels, not just at the interaction between high- and mid-level representations as in this implementation.

The difficulty in implementing low-level alignment in my effort to build a vision system using propagators sheds light on some obstacles in the way of applying the full power of the propagator architecture in a vision context. In order to test the alignment hypothesis, it is necessary to align at all levels: to propagate high-level knowledge through intermediate representations and down to low-level representations. The propagator architecture was an appealing choice of mechanism for pervasive alignment, because its graphs of components that communicate through shared state, combined with its ability to make partial solutions available quickly, are both desirable attributes for a system that aims for pervasive alignment of partial information across sensory modalities working toward a common perceptual goal. Having a network of components communicate through state shared only with neighbors is appealing because it permits pervasive alignment while only requiring system designers to build local interfaces between components. Early propagation of partial information allows maximum benefit from alignment: a component can, for example, opportunistically reduce the domain of a state variable it shares with a neighboring component as that neighboring component continues an ongoing process, thereby potentially ruling out alternatives before significant work has been invested. Despite such appealing properties, I believe that the propagator architecture and its basic components as originally specified do not constitute a complete framework for implementing alignment-based vision systems. I present here my assessment of the main obstacles to development of vision systems with propagators, followed by my thoughts on how to overcome the obstacles.

Reflecting on my work with propagators reveals two main areas of difficulty: a technical type and a conceptual type which has several subcategories. The technical type is not a fundamental limitation but it does present a formidable engineering challenge. Because propagator frameworks are relatively new and under-explored, there are very few existing primitives available with which to build propagator-based vision systems. There is, however, an extensive body of work on computer vision that continues to grow at a rapidly accelerating pace. A daunting task thus confronts those who venture to rebuild the basic machinery that has driven recent progress in computer vision, in such a way that it can take full advantage of propagation. Even the simple example presented in this chapter contains many components that derive from software libraries, which cannot benefit directly from propagation without substantial design effort. For example, consider optical-flow calculation. It is surely feasible and likely rewarding to incorporate top-down information in optical-flow calculation. State-of-the-art optical flow methods already use high-level image features (Revaud et al. [2015]) as well as deep feature learning (Weinzaepfel et al. [2013]) to produce ever more accurate optical flow vector fields. Incorporating top-down information, such as classes of objects like pedestrians, or likely activities such as walking, may therefore have tremendous potential to improve upon existing optical flow methods. Implementing an optical flow calculator that can accept top-down information, however, would be a substantial undertaking worthy of its own dedicated research program. The optical flow library provided by Adarve and Mahony [2016] that I used in my implementation consists at the time of this writing of 25,782 source lines written in a combination of C++, CUDA C++, ANSI C, and Python. Much of the project relies on the the authors' knowledge of and careful attention to the architectural minutiae of a certain brand of GPU, but even neglecting the special hardware expertise required by the project, a Basic COCOMO model (Person-Months $= 2.4N_{KSLOC}^{1.05}$) (Boehm et al. [2000]) es-

timates that the project required over 6 person-years of effort[2]. Technical problems abound of a magnitude equal to or greater than that of building a propagation-enabled optical flow calculator, and so any strategy requiring reimplementation or deep modification is clearly infeasible. In order to build propagator systems that perform on par with the state of the art, therefore, it will be very important in future efforts to invent ways to include and combine many opaque techniques in propagator architectures.

I identified 4 categories of conceptual obstacles that I believe are fundamental to the propagator architecture that I used in my work when it is applied to low-level visual processes. It is useful to organize the obstacles this way, because doing so has helped to circumscribe the aspects of the propagator architecture that appear problematic in order to retain as much of the architecture's power as possible in the work that is the subject of Chapter 5. The obstacles discussed here apply to the propagator architecture of Radul [2009] upon which I based my work, and throughout the remaining discussion I use *propagator architecture* to refer to my implementation based on Radul's groundwork. Although generalizations and variants of the architecture exist which are unaffected by these particular obstacles, it is important to note that the obstacles arise from several of the great strengths of this chosen architecture, which I identified in Chapters 1 and 3 and which led me to use the architecture in my work. These obstacles do not detract from those strengths under many circumstances; only under some circumstances pertaining to low-level vision. Careful review and consideration of the obstacles will enable, in future efforts, unification of the original architecture with its variants that are unencumbered by the obstacles. Such unification will allow high-level propagation systems like the one that is the subject of this chapter to work with systems that appropriately model low-level vision. The 4 obstacles are as follows:

1. **Scarcity of strong constraints**

   Scarcity of strong constraints refers to an apparent scarcity of constraints of the type that the propagator architecture can make use of, in problems of interest in low-level vision.

2. **Brittleness of logical absolutes**

   Brittleness of logical absolutes is the problem that, even when strong constraints of the type that propagation can use are present, they tend not to provide much in the way of actionable information to the inherently goal-directed processes of vision.

3. **Incorrigibility**

   Incorrigibility refers to the inability of propagation to be corrected once it has made its decision, which leads to certain problems that are not present in systems that can revise decisions.

4. **Problems of scale**

   The problems of scale are that primitives in the propagator architecture suffer a substantial reduction in efficacy when they are forced to cope with high-dimensional data.

I discuss each of these issues, and conclude with a discussion of how to address them while preserving some of the unique benefits of the propagator architecture, in order to come closer to a vision system that realizes the goal of pervasive alignment.

[2]Estimate generated using David A. Wheeler's 'SLOCCount'.

4.4.1 Scarcity of strong constraints

When strong constraints are scarce, it is unproductive to propagate information by domain reduction. In the problem of map coloring, for example, propagation alone cannot arrive at a coloring that guarantees that no adjacent regions on a map share a color. Map coloring is a convenient analogy in this context because, like vision, map coloring is a problem without a uniquely-determined solution. Map coloring is also goal-directed: any coloring satisfying the constraints will do. Analogously, vision systems exist to serve the needs of survival and higher cognitive function, not to find all possible interpretations of their input or even globally-consistent interpretations. The propagator architecture is well-suited to exploiting *strong*, logical constraints. It seems to me that weak constraints, of the type that can be expressed as probability distributions, abound in low-level vision, but strong constraints are very scarce. In Section 4.3.3 I described ways to treat soft constraints like hard constraints: by thresholding, and by using cells that represent changing beliefs that nevertheless only accumulate information, either by accumulating factors of a distribution's parameters, as in the symmetric cascade, or by accumulating discrete examples in growing histograms. While these methods work decently at the intermediate levels of representation used in object tracking, it is problematic to apply the methods at the low level of, for example, background color modeling, where the combinatorics are unfavorable. Strong constraints in low-level vision thus seem too scarce to be of any use.

4.4.2 Brittleness of logical absolutes

Separate from the problem of finding strong constraints is the problem of learning something valuable from propagating those constraints. Reasoning in terms of logical absolutes is conducive to building propagator machinery with useful guarantees, such as idempotence. Logical absolutes also enable powerful ideas: dependency-directed backtracking (Stallman and Sussman [1977], Zabih [1988]) forms nogood sets based on observed logical inconsistency, and as a result it can efficiently prune search graphs. Without a strong notion of impossibility, dependency-directed backtracking would be reduced to, at best, a heuristic search e.g. Monte Carlo tree search. Despite its irrefutable superiority in many inference scenarios, reasoning in terms of logical absolutes also has an inescapable brittleness which, it seems to me, precludes its productive use in low-level vision. Essential measures of optimality may be overshadowed by logical possibility, so that a reasoning framework that must retain all possibilities can make no progress. Whenever absolute impossibility can be found in the noisy and imprecise world, it is isolated from nearly all variables on which it could have any practical effect.

The issue at hand when considering the brittleness of logical absolutes is not whether observations and inferences about the visual world can be directly represented by logical relations. A large body of work from AI's early history demonstrates the infeasibility of that approach. I considered instead whether the primitives of the propagator architecture, which operate in the realm of logic, could be applied to model and productively reason about empirically-measured probability distributions. One way of doing this is to have cells that store information about the parameters of parameterized probability distributions[3].

[3]Another way to accommodate probabilistic weak constraints in the propagator framework is to accumulate individual samples in dedicated cells. This way shows promise in reducing brittleness and in resolving other problems as well. I discuss it further in Section 4.4.5.

For example, consider what it would take to overload the `sum` propagator to operate on probability distributions. It is useful to think about the distinction between *sample space* and *distribution space*. Suppose two cells store information about independent normally-distributed random variables X and Y. That is, one cell stores the respective ranges of values that μ_X and σ_X^2 may hold, and the other cell stores the ranges of μ_Y and σ_Y^2. Suppose we are interested in the sum, $Z = X+Y$. In sample space, the `sum` operates on x, y, and z, samples drawn respectively from \$X, Y,\$ and Z. The familiar sum relation holds, and given interval constraints on any two of the variables the `sum` can uniquely determine the interval constraints on the third. In distribution space, we are propagating information about the parameters of the distributions of \$X, Y,\$ and Z. To do this we need additional knowledge of dependence structure. Given the dependence structure and other particulars of of this example, namely that $X \perp Y$, $X \sim \mathcal{N}(\mu_X, \sigma_X)$, $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$, and $Z = X + Y$, our distribution-space-overloaded `sum` should propagate according to the relations $\mu_Z = \mu_X + \mu_Y$ and $\sigma_Z^2 = \sigma_X^2 + \sigma_Y^2$.

Propagation of distribution parameters in isolation may prove useful for describing complex probability distributions, but this is only part of the problem we need to solve. The good news is that when we know not only the conditional independence structure of a graph of interconnected random variables, but we additionally have detailed knowledge of the *computational dependencies* of the internal variables of the distribution represented by the graph, many possibilities open up for reasoning in terms of the probability distributions represented by programs with nondeterministic elements. In this way, such distribution-space propagation is similar to probabilistic programming (Goodman et al. [2012]). Unfortunately, to apply distribution-space propagation and its useful generalization capacity in vision problems of interest, there needs to be a way to update the distribution-space parameters from sample-space measurements. There seems to me to be no good way to do this within the propagator framework, as no amount of sample data can ever provide incontrovertible evidence to support pruning the domain of distribution parameters such as the means and variances in the $Z = X + Y$ example. In general, it is impossible to guarantee that the interval bounds of the distribution parameters that we estimate from an open-ended collection of samples would change monotonically. If the system cannot move from sample space to distribution space it cannot easily generalize and it is stuck reasoning in terms of an ever-growing set of specifics. The best hope for resolving this issue within a strictly-logical framework is to generalize over finite batches of examples at a time. I discuss this strategy in Section 4.4.5.

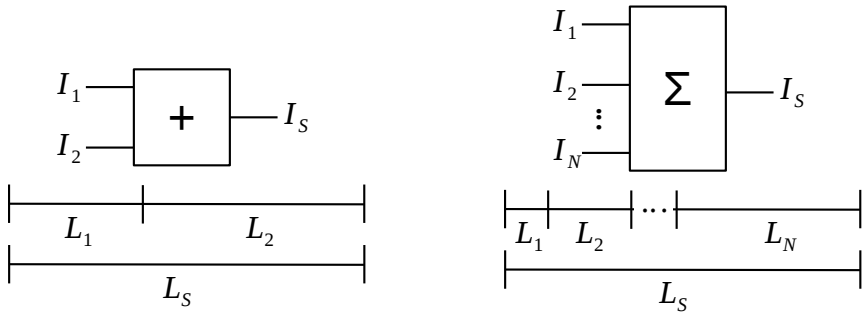
4.4.3 Incorrigibility

Saying that propagators sometimes behave incorrigibly calls attention to a negative aspect of one of their greatest strengths: monotonicity. Monotonicity empowers propagator systems in several ways. It allows propagators to begin making use of each other's work during ongoing computations; they can rely on the partial results because of the guarantee that change cannot be undone. It serves as a crucial building block of great properties for networks to have, such as convergence guarantees. Unfortunately monotonicity also means that propagator networks cannot be corrected without retraction of premises, which suffers from unfavorable combinatorics in many problems of interest in low-level vision. Truth Maintenance Systems (TMS) (McAllester [1978]) allow propagator networks to identify and reason about conflicting premises, but in a TMS conflict is absolute; degrees of conflict cannot be represented in terms of a continuously-varying cost function. Incorrigibility makes it cumbersome at best to implement relaxation algorithms in the

propagator architecture, especially the type of relaxation algorithm in which the cost of violated constraints is reduced by repeated local adjustments. Having identified relaxation algorithms as a crucial family of algorithms in Chapter 1, I believe incorrigibility is a significant obstacle that needs to be addressed.

4.4.4 Problems of scale

Certain foundational components of the propagator architecture used in this work make heavy use of interval arithmetic. As a result, the components suffer from precipitous degradation in their ability to propagate domain reductions as the dimensionality of their input-output space increases. To see why, first consider the motivating thought experiment used by Sussman and Radul [2009]: measuring the height of a building using a barometer. In their thought experiment, the authors considered several ways to obtain measurements: throw the barometer off the roof of the building and time its fall, compare the shadow cast by the barometer to that cast by the building, and bribe the building superintendent with the barometer to obtain an estimate of the building's height. The authors implement each of these methods in terms of a small number of primitive propagators, for example `quadratic`, `product`, `sum`, and a handful of other relations. I focus on the `sum` relation, and argue that the outcome applies to other propagator primitives as well.



(a) 3-way sum propagator (b) Multi-way sum propagator

Figure 34: A three-way and multi-way `sum` propagator

`sum` propagators connect to two (a) or more (b) summands, and their respective totals. If the summands are unbroken intervals, then the total I_S is also an unbroken interval, and its length L_S is equal to the sum of the lengths of the summand intervals, $I_1 ... I_N$. The thresholds on the lengths of the sub-intervals of I_S in which propagation can occur lead to unfavorable scalability issues.

The barometer example makes use of a `sum` propagator like that depicted in Figure 34a, with three connections: two summands and a total. Suppose that such a 3-way `sum` has its summands connected to two cells each containing the interval $[0, 1]$. The cell connected to the total contains the value $[0, 2]$. This is a stable configuration; no pruning of the intervals can be performed based on the relation enforced by the `sum` propagator. Now consider what happens when the total is updated, that is, pruned by way of some external update, so that it contains the interval $[1, 2]$ instead of the previously-held $[0, 2]$. Note that the `sum` propagator can do nothing to change the domains of either summand. A value of 1, the lower bound of the new total interval, can

be realized if one summand has value 0 and the other has value 1. Now suppose we further restrict the total cell, to contain the interval $[1 + \epsilon, 2]$. Fortunately, now the `sum` propagator can do something: it may update the domains of each summand to hold the interval $[\epsilon, 1]$. In other words, in order to get propagation in this example by restricting the total to be no less than some value T_H, it must be the case that $T_H > 1$. If we instead restrict the total to be no greater than some value T_L, then propagation can only happen when $T_L < 1$. Another way to think about this example is that if we pick a random value in the interval $[0, 2]$ and then decide at random to eliminate values less than the chosen value or higher than it, the `sum` propagator will get to do some propagation half of the time.

Being able to propagate some information half of the time might not seem that bad. Now consider what happens in the case depicted in Figure 34b, in which the `sum` propagator has N summands and a single total. Assume that the summands and total are all finite unbroken intervals of non-zero length; this simplifying assumption doesn't change the implications to scalability. Likewise assume that, as in the last example, the system starts in a steady state, in which the total interval is constructed solely based on the information supplied by the summand intervals. There are N summand intervals I_j, each described by the following bounds:

$$I_j \equiv [l_j, h_j] \tag{8}$$

That is, l_j and h_j are the respective lower and upper endpoints of the summand interval I_j. Let L_j be the length of the summand interval I_j:

$$L_j \equiv h_j - l_j \tag{9}$$

And let the intervals be listed in order of increasing length, that is:

$$L_1 \leq L_2 \leq ... \leq L_{N-1} \leq L_N \tag{10}$$

The total interval, I_S, that can be inferred directly from all summand intervals is:

$$I_S \equiv [l_S, h_S], \quad l_S = \sum_{j=1}^{N} l_j, \quad h_S = \sum_{j=1}^{N} h_j \tag{11}$$

And the length of the total interval is L_S, which is the sum of all the lengths of the summand intervals:

$$L_S = \sum_{j=1}^{N} L_j \tag{12}$$

With these definitions, we can now answer questions about when pruning of the summand intervals can happen by propagation of an update to the total interval. Specifically, suppose the total interval I_S is shortened by picking a random value from a uniform distribution over the interval, separating the interval at the randomly chosen value into two new intervals (each inclusive of endpoints), and retaining one of the two resulting intervals at random. The question to answer is: what proportion of the time would *any* propagation happen? That is, when would any of the summands' intervals change? There are other interesting questions to ask as well, such as how much total length do we expect to be trimmed from all of the summand intervals, or which intervals become trimmed under which conditions. The answers to those questions, along with detailed supporting analysis for the results that follow, are straightforward but omitted for clarity. The lower bound of any summand interval cannot increase unless the lower bound of the total exceeds a threshold T_H:

$$T_H = l_S + \sum_{j=1}^{N-1} L_j = h_S - L_N \tag{13}$$

No lower bound of a summand interval can increase before the lower bound of the largest summand interval increases[4]. The lower bound of that largest summand interval cannot increase until the length of the total interval is forced by the rising lower bound to be less than the length of the largest summand interval. In an analogous way, the upper bound of any summand interval cannot decrease until the upper bound of the total interval falls below a threshold T_L, which must be as follows:

$$T_L = l_S + L_N \tag{14}$$

The probability p_T of any propagation occurring given the previously stated condition, a random split of the sum interval, is then:

$$p_T = \frac{L_N}{L_S} = \frac{L_N}{\sum_{j=1}^{N} L_j} \tag{15}$$

If the length of the largest interval stays fixed this indicates that p_T is strictly decreasing as the number of inputs grows. In the common scenario where the starting domain of all summands is identical, this means that that p_T, **the probability that the `sum` propagator can do *anything at all* in response to a domain reduction, is inversely proportional to the number of inputs**. This result is especially bad considering that a summation with thousands or millions of inputs would not be out of the ordinary in a typical low-level vision problem; for a general sense

[4]The intuition follows from the fact that all summands contribute to the total in identical ways. If the largest summand interval retains its entire capacity, we can always accomodate a small change in the total using a value in the interval of this largest summand before we must resort to limiting another variable.

of scale, an RGB video at 640x480 resolution and 30 frames per second contains about 27 million individual measurements per second.

The problem comes from treating all of the summands independently of one another. This hints at why the problem is not limited to sums. A `product` propagator is almost the same machine as a sum with `log_exp` propagators sitting between its cells and the rest of the network (neglecting the 0 case). In fact any differentiable[5] operation that has the property that it takes $N-1$ of its input/outputs to uniquely determine the last one has the scale problem I described. It is not true that the constraint which causes no pruning in the propagator scenario conveys no information. Figure 35 depicts where the information from such a constraint exists: in a hyperplane in the space of the summands, represented by the diagonal lines in the figure. The scalability problem can happen whenever the constraint conveys information that can only be represented in some subspace of the inputs with dimensionality greater than 1. To fix the scalability problem, suppose we introduce a new kind of cell to the propagator system that stores constraints as manifolds in the input space. As it turns out, this new kind of cell would have no trouble with the N -ary `sum` propagator. Any pruning of the total would result in some pruning of the hypervolume represented by the summands.

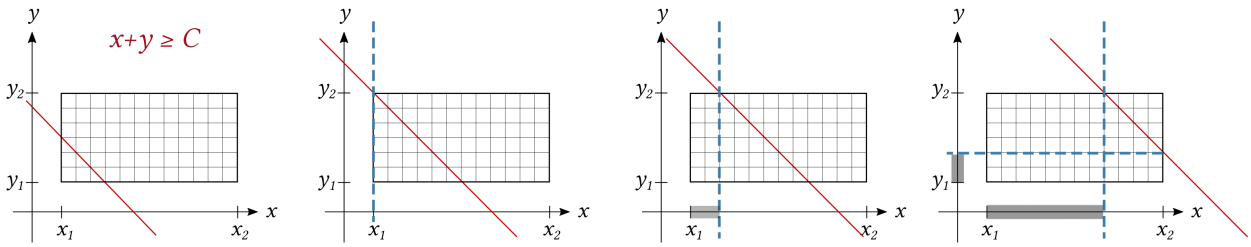


Figure 35: A three-way `sum` propagator with a constraint on the total
As the value of the constraint C increases, nothing can be pruned until C surpasses y_2+x_1. The interval of y cannot be pruned until C surpasses x_2+y_1. The gray part of the rectangle is invalid under the constraint. The grayed regions along the axes can be eliminated by the `sum` propagator.

The surprise is that, with the addition of this new type of cell that represents constraints as manifolds in high-dimensional space, the resulting hypothetical propagator network looks conspicuously like a neural network. This is great news, because it hints at a way that a variant of the propagator architecture can connect seamlessly with neural networks. I revisit this idea in Chapter 5.

4.4.5 Where to go next

The approach I have taken to resolving the issues of scarcity of strong constraints, brittleness of logical absolutes, incorrigibility, and problems of scale is to sacrifice monotonicity and have cells store values that can change. With this modification, it is possible to adjust variables' values individually based on their contribution to the cost of a violated high-dimensional constraint. Because this is how the weight variables in neural networks are trained, it can make the network

[5]Differentiability, along with the other property stated of not distinguishing between inputs, are sufficient criteria. It seems plausible that more inclusive criteria exist.

compatible with neural networks if they also have certain other properties, like differentiable operations and cost functions. I revisit this in depth in Chapter 5. The sacrifice of monotonicity precludes strictly logical inference about variables' values based on the constraints; logic gives way to optimization, and one or more cost functions must determine how to bring the variables into alignment with the constraints. Perception problems typically have objectives that are difficult to optimize, which presents a significant challenge. Other, more interesting challenges abound: for example, as I discuss in Chapter 5, this marriage of propagator-like machinery with neural networks enables a type of hypothesis testing that is not possible in ordinary neural networks.

An interesting direction to pursue in future work on applying the original monotonic propagators framework to problems with a lot of data is to develop further the type of cell that accumulates information by way of examples added to collections. This type of cell, which I used to store information about occluders by representative examples as discussed in Section 4.3.3, manages not to sacrifice monotonicity and addresses the problem I introduced in the discussion about the brittleness of logical absolutes: that it is problematic to estimate the parameters of probability distributions from samples in a way that honors monotonicity. One way to interpret monotonicity of information content is to use a type of cell that contains a growing set of representative examples. For instance, one such cell could contain data points, and another cell could contain statistics like means and variances of batches of data points from the first cell. An `estimator` propagator sitting in between would measure statistics of batches from the data cell, and submit updates to the statistics cell. The `supported-by` property of the statistics updates would derive from the `supported-by` properties of each corresponding batch of data points. A `sampler` propagator could watch for updates to the statistics cell, and submit samples from the corresponding distribution (assuming known distribution type) to the data cell. The `supported-by` properties would prevent echoing, that is, the undesirable situation where the `estimator` operates on the output of the `sampler`.

Set membership is a logical constraint and so this arrangement doesn't have to break the rules. The advantage that monotonicity imparts is somewhat diminished from that of other logical constraints; in this case it is relegated to controlling the flow of information. Items already in a set (as established by `supported-by` properties) need not be recomputed. This is a powerful idea, though, because it allows the flow of information to be determined by the needs of the problem rather than explicitly by the programmer.

4.5 Contributions

My main contributions in this chapter are as follows.

- I implemented a propagator system that tracks pedestrians in a video from a stationary camera, in order to demonstrate how to use information about the geometry of certain objects and their motion to make inferences about geometry and motion of other objects. The system shows one way in which alignment can be applied to a vision problem.

- I introduced the symmetric cascade, an inference method that benefits from the power of propagation. The symmetric cascade is a general-purpose mechanism that can efficiently confirm or invalidate hypotheses. It can be optimized to make situation-dependent tradeoffs of accuracy for efficiency, and it can be learned from training data.

- I discussed four obstacles to development of low-level vision systems with propagators: scarcity of strong constraints, brittleness of logical absolutes, incorrigibility, and problems of scale. The discussion motivates the decision to remove the monotonicity property and continue development with a modified propagator architecture that is compatible with neural networks. Chapter 5 details that work.

5 Building Neural Networks for Alignment

In this chapter you learn about my steps toward alignment-driven neural network architectures. A significant outcome of the effort is that alignment architectures enable a neural network trained for one task to accomplish another. As shown in Figure 36, the neural network trained to estimate depth from a single image can also perform the related task of depth super-resolution, that is, using an image to increase the level of detail in a supplied low-resolution depth map.

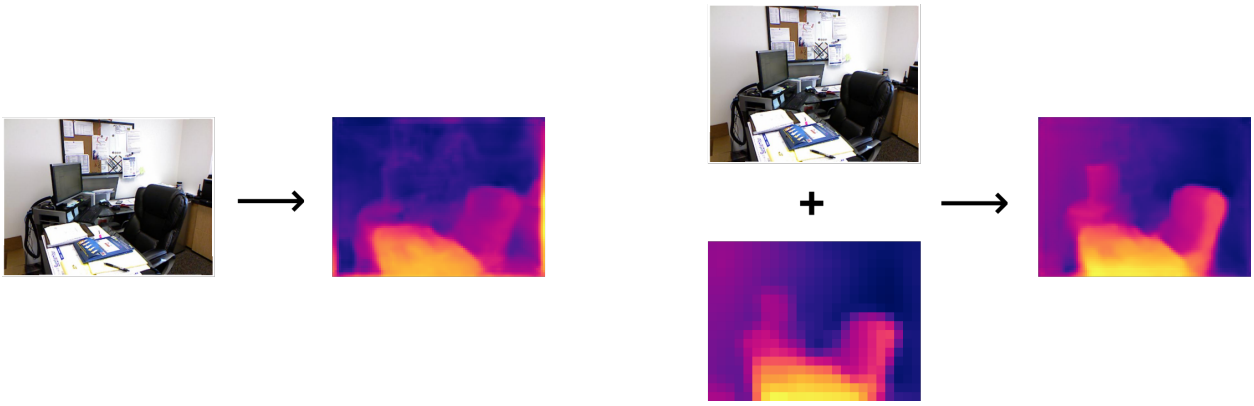


Figure 36: Dual-mode estimation neural network

The neural network architecture presented in this chapter learns to estimate depth from images (left). Its design enables it to perform the task of depth super-resolution (right) without retraining.

5.1 Introduction

The central endeavor of my work is to build a perception system that exhibits pervasive, multimodal alignment. The system of propagators that is the subject of Chapter 4 demonstrates at a high level how propagators can be applied to making inferences in a scene-understanding context. As implemented, the system is limited to using propagators at a coarse level; the low-level components of the system are external to the propagator architecture and therefore cannot benefit from it directly. In order to take propagator systems in vision to the next level of performance and robustness, it is necessary to build low-level components that can exploit the full power of the propagator architecture. By allowing bidirectional information flow to propagate farther down in the representational hierarchy, the influence of environmental and multimodal constraints can extend to the interpretation of low-level visual features.

In Section 4.4 I discussed several outcomes of that effort to build alignment systems with propagators. One outcome was an observation that with modification to support multidimensional constraints, certain components of the propagator architecture begin to resemble units of artificial neural networks. In this chapter I describe my steps toward building neural networks that can serve as building blocks for robust multimodal perception systems.

It has become a fashion to begin machine-learning articles with ecstatic praise for deep neural networks. *Deep convolutional neural networks have become the staple of the modern computer vision pipeline, have pushed performance to soaring heights on a broad array of problems, are extremely*

versatile, driving unprecedented advances in segmentation, detection, recognition, localization...[6] Of course, these accolades are justified in the sense that they consist solely of true statements. I believe, though, that the explosive growth of this technology elicits forbearance as much as it inspires respect and awe. Deep learning, as a methodology that is rapidly developing into a discipline of its own, has largely cast aside one of the greatest promises of AI: to uncover the principles underlying human intelligence. Calling back to the physics metaphor I cited in Section 1.3, deep learning in its present state shows little hope of revealing anything as profound as the $\boldsymbol{F} = m\boldsymbol{a}$ of intelligence. As a discipline, deep learning is more concerned with replicating behavior than accounting for it in any more abstract terms than the meager abstractions of statistics. The issue that is the subject of Chapter 2, that deep neural nets rely upon features for classification that are often conspicuously irrelevant to a human understanding of the visual problem being solved, points to a problem that should concern engineers and scientists alike.

Perseverance in seeking out the principles and computational imperatives of intelligence will eventually outpace the bubble of enthusiasm over deep learning's rapid overturning of present-day AI benchmarks. Understanding intelligence's underlying principles will be scientifically rewarding and, I anticipate, necessary to overcome tomorrow's engineering challenges. Just as important to recognize is that the profound achievements of deep learning have secured its place among the most valuable tools that we can apply in the endeavor to understand intelligence. These motivations guide my work with deep neural networks and frame the work I describe in this chapter. In the following sections, I introduce the depth estimation problem of interest. I discuss related work on similar problems and point out differences in approach between existing work and my own contributions. I motivate and summarize the architectural details of my network. I present experimental results and discuss opportunities for improvement.

5.2 Problem statement

To motivate development of neural network architectures for multimodal alignment, I focused on the problem of predicting the distance from the focal plane (*depth*, when unambiguous) of the physical surface that best corresponds to each pixel in a single RGB image. This problem fits well with the theme of geometric inference that motivated my work in Chapter 4, and has some additional advantages:

1. Depth reconstruction from a single monocular image is underdetermined. Though the problem is impossible to solve in the general case, humans easily identify the relative and approximate depths of objects within an image, demonstrating that we must use context and domain knowledge to solve this problem in practice.

2. Ground-truth depth values are easy to obtain from a wide variety of sensors. Publicly-available datasets abound.

3. As a non-visual channel, depth maps are analogous to the information provided by another sensory modality such as touch. Depth thus presents the opportunity to test the multimodal aspect of the alignment hypothesis. Specifically, can a system learn a robust visual representation by learning the cross-domain problem of estimating depth?

[6] Paraphrase comprised of excerpts from Donahue et al. [2016], Chen et al. [2014], Pathak et al. [2016], Shelhamer et al. [2016], Girshick [2015], Sermanet et al. [2013].

4. Unlike many non-visual channels, depth maps conform conveniently to the shape of images, having depth values corresponding to pixel locations within an image. The fact that depth maps are 2D maps with image correspondence is convenient for many reasons. Of special interest here is that it makes them conducive to processing with convolutional neural networks.

5. Visual depth estimation is a useful ability in and of itself. Robust autonomous systems should be able to estimate depth from vision alone, as we can, so that they can perform safely when their other means of sensing depth, such as LiDAR and structured light sensors, fail them.

6. Unlike measurements from other sensory domains, such as hearing or smell, depth is easy to simulate faithfully in artificial environments.

There are clear limitations to this problem as well, that make it less ideal than related problems as a testing ground for models of visual intelligence. Experiments on human subjects suggest that monocular cues such as angular declination enable accurate reasoning about 3D object position (Loomis [2001], Ooi et al. [2001]) but experimental limitations restrict these results to objects within about 7.5 meters of the observer. Even if it were clear that humans are capable of accurate monocular depth perception over a broad range of distances, it is a conceptual mistake to assume that it is feasible or appropriate to learn how to perceive depth from images without additional information. It is likely that humans bootstrap complex spatial reasoning from many different kinds of experiences, and as a result we are able to infer depth from images fairly well. Training a system to output depth from images in a supervised-learning setting with just those inputs may result in a similar type of brittleness to that which the phenomenon of adversarial fooling exposes in neural network image classifiers. It is important, therefore, not to mistake good performance of systems trained this way with *robust* performance until they demonstrate attributes characteristic of robust systems. Exhibiting strong generalization to environments substantially unlike those used for training, for example by training on real images and testing on line drawings, is a better indication of robust performance than the typical view of generalization to a held-out set from a fairly homogeneous dataset. Of course, if adversarial examples are found, they must be regarded as strong proof that the system has not achieved robust performance.

Despite the limitations of the problem of learning to estimate depth from single images, I believe it is a good first step toward building multimodal alignment systems. Due to the substantial complexity of working with deep neural networks and their associated infrastructure, the simplicity of this problem's scope adds to its appeal. Expansion to more complex problems, thus enabling more robust models, is an exciting area of future development that I discuss in Section 5.7.

5.3 Related work

There is a large and rapidly accumulating body of work on the subject of depth estimation in general, and depth estimation from single monocular images in particular. I focus on a few illustrative examples in order to point out salient differences between my approach to this problem and the approaches used by others.

The Make3D program developed by Saxena et al. [2008] uses an MRF to reconcile surface positions and orientations of the approximate surfaces represented by neighboring superpixels of an image. The authors used an engineered feature set, based on such monocular cues to depth as sizes of detected objects, texture gradients, and line orientations, as features in a supervised learning setting. Hoiem et al. [2005] (HEH) obtained similar results with a simpler, cut-and-fold model of transforming images to 3D models. HEH used a learned classifier to group superpixels into constellations belonging to ground, vertical surfaces, or sky based on a set of 66 engineered features. Both Make3D and the work of HEH, as well as similar more recent work on estimating depth from object labels with engineered features by Ladický et al. [2014], produced qualitatively good results, especially considering that their work preceded the popularity explosion of deep feature-learning. This work is also task-specific in that the engineered features and the learned models are designed solely to estimate depth from monocular images. By contrast, my interest in depth estimation in a feature-learning context is in using depth estimation to help the system learn the regularity and constraints of the physical world, thus producing more robust learned feature representations than it would learn if it were trained on a purely visual task.

Eigen et al. [2014] and Eigen and Fergus [2014] used staged convolutional neural networks to estimate depth from images. The first stage of the architecture of Eigen et al. [2014] estimated a low-resolution depth image from the RGB image, which their system then presented along with the original RGB image to a second stage, which produced a higher-resolution depth image. The work of Eigen and Fergus [2014] expanded this architecture by adding a third stage, and by passing feature maps between stages as I did in my work rather than passing depth images directly. Furthermore, Eigen and Fergus [2014] expanded the original work to a multi-task estimation problem, estimating surface normals and semantic labels alongside depth.

In my work I likewise used a staged architecture with interpretable internal signals. A mechanistic difference between my work and the work of Eigen et al. [2014] and Eigen and Fergus [2014] is that my network is fully convolutional whereas the networks in the referenced work require several fully-connected layers, and as a result my model is significantly smaller than that of Eigen and Fergus [2014] while producing qualitatively good results, and my model scales, without modification, to inputs of different sizes. Also unlike this referenced work, my architecture reuses weights among the stages via skip connections, drawing on intuition that the same features that build up a hierarchical representation leading to approximate global depth estimation can also contribute to refining that global depth estimation. I discuss this intuition in Section 5.4 and illustrate it in Figure 37. Most importantly, my focus is on achieving new ability through semantically-interpretable ports rather than achieving benchmark milestones. My architecture derives unique benefit from the interpretability of the internal signals through a propagator-like mechanism to inject signals from other sources into the neural network. I discuss this in depth in Section 5.6.5.

Li et al. [2015] use learned features transferred from AlexNet (Krizhevsky et al. [2012]) for depth estimation via conditional random field (CRF), thus demonstrating the potential for features learned for image classification to inform depth estimation. Liu et al. [2016] take the merger of neural networks with graphical models a step further, by exploiting continuity assumptions and differentiable potential functions in the graphical model to train their neural network to learn the unary and pairwise potential functions of a CRF. Laina et al. [2016] show that a conceptually simpler fully-convolutional model based on the architecture of ResNet (He et al. [2015]), albeit with a specialized reverse-Huber cost function, also performs well. Kuznietsov et al. [2017] build

further upon the ResNet-based approach with a specialized cost function that incorporates stereo disparity and sparse ground truth obtained from LiDAR to train the neural network.

I aim to take the thematic result of such work, that feature representations learned by convolutional networks perform well on depth estimation, to the next level by showing that a multi-modal, multi-task system with flexible, task-dependent data-flow can improve robustness simultaneously on many tasks. The salient difference between my work and the related work on monocular depth estimation with neural networks cited here is that, where the authors of the cited work focus primarily on advancing the state of the art on monocular depth estimation as an isolated problem of interest, I seek to understand whether incorporating depth estimation in multi-task perception systems improves robustness of those systems, by forcing the systems to learn more relevant representations.

Apart from the push to advance the state of the art in monocular depth estimation using ever deeper and higher-capacity neural network models, some recent work in the area has developed interesting, perceptually-motivated solutions to the problem. Zoran et al. [2015] use the insight that sparse pairwise relationship estimates, such as *point A is farther than point B*, are easier to obtain than dense metrics such as maps of absolute depth. These authors' depth-estimation framework comprises a method of extracting point pairs from an image, a three-way classifier for assigning ordinal relationships to the points, and a method to propagate the resulting partial ordering of points to form a dense depth map. This framework has strong potential to combine with the work that I present in this chapter, because the explicit use of propagation may allow it to complement methods I have developed for depth estimation. Specifically, my neural network depth estimation models are designed to accept signals from external sources to enhance their own predictions. The method of Zoran et al. [2015] seems to be a compatible external information source, specifically because it generates sparse depth maps and the models I developed can accept sparse depth maps and increase their resolution. Thus a promising area for future development would be to connect the frameworks together.

In work by Zhang et al. [2016b], the authors framed the problem of depth estimation from monocular images as an unsupervised learning task in which a variant of an autoencoder (Hinton and Salakhutdinov [2006]) learns two disjoint prediction pathways in order to reconstruct its input: the RGB-D images recorded by a structured-light sensor. One pathway sees only the depth channel and must reconstruct the image, the other sees the image and must reconstruct the depth channel. The authors argue and present evidence that their *split-brain autoencoder* architecture learns a stronger feature representation than autoencoder architectures that rely on representational bottlenecks, by forcing cross-modal generalization rather than compression of the input. This is the same intuition that Coen [2006] first identified and supported with extensive experimental results, and which motivates my work as well: that redundancy of the senses empowers self-supervised learning of the regularity and constraints of the natural world. My focus is on how best to apply this intuition to build robust perception systems.

5.4 Approach

In this section I describe the neural network architecture I used to estimate depth from monocular images. I first motivate the design choices underlying the architecture in terms of my overarching goal of building alignment-driven vision systems, and in consideration of observations about how neural networks achieve good statistical performance while remaining susceptible to brit-

tle failures, as demonstrated in Chapter 2. Breaking with the tradition of *just so* motivations of the design choices guiding neural network architectures, I motivate my design choices in terms of three main desiderata: that the network should have fortifications against adversarial examples, that it should have semantically meaningful ports, and that it should build upon existing, empirically-successful designs. Additional less prominent design decisions that arose from iteration of the design are identified as such, with justification in terms of observed shortcomings and desired improvements. I discuss each of the main desiderata here.

5.4.1 Fortification against adversarial examples

A high-level lesson from the results of the work described in Chapter 2 is that neural networks take the path of least resistance. The apparent ubiquity of adversarial examples, and the relative ease by which they are generated, hint at a deep problem with deep feed-forward neural networks: it may not be safe to assume that solutions to many problems of interest can be fully described by continuous or piecewise-continuous manifolds in some representation space. This assumption underpins most, if not all, work with neural networks. We should be suspicious that there might always be strong counterexamples, that is, anomalies that have no sensible interpretation in terms of natural phenomena, that are arbitrarily close to normal examples, e.g., correctly-classified images. If that is the case, no architecture within the current feed-forward paradigm will be immune to fooling by adversarial examples. Ben-Yosef et al. [2015] suggested that in natural vision, bottom up recognition stages similar to high-performing neural network models trigger activation of top-down processes to provide the detailed compositional interpretation observed in natural vision. A generalization of this idea is that feed-forward processes provide *hypothesis generation* which must be complemented by *hypothesis testing* via feedback processes. One form of hypothesis testing is the type explored by Ben-Yosef et al., namely, evaluating a recognized object to identify its components. I anticipate that such hypothesis testing via feedback will be the only reliable way to fully harden a vision system against adversarial examples and related brittle phenomena.

Aside from widespread neglect that adversarial fooling is currently a serious impediment to development of robust vision models, one reason that feedback is not more prevalent in vision models is the significant technical challenge of training models with feedback via stochastic gradient descent (SGD). Training networks with feedback via backpropagation through time leads to problems of exploding or vanishing gradients. As shown by Pascanu et al. [2012], if all eigenvalues of the recurrent weight matrix are less than 1, all sufficiently long-term components of the gradient will vanish. If at least one of the eigenvalues of the recurrent weight matrix is greater than 1, then it is possible for the gradient to explode. Rather than tackle head-on the formidable optimization challenges inherent in applying feedback to neural networks, I used a compromise strategy to help fortify the network against adversarial fooling, while preserving a feed-forward architecture.

The strategy I used to help fortify the network while preserving a feed-forward architecture is to enforce several different cost functions at different output ports within the network, to encourage the network to develop more relevant feature representations. End-to-end training of deep networks has led to demonstrably better feature representations than the engineered feature representations of earlier models. As the experiments in Chapter 2 suggest, though, the huge capacity of deep models may lead to a subtle kind of overfitting in which networks develop

good measured generalization performance on the task used for training, e.g. 1000-way image classification, but they do so by means of *shortcut* feature representations that do not stand up to scrutiny and are easily fooled. One way to mitigate the shortcut-finding problem is to engineer several cost functions to enforce good attributes of a feature representation, without engineering the representation itself. This is the approach I have taken. I note that small patches of images provide relative-depth clues, like occlusion and shadows. Intermediate-size patches may provide some absolute-depth clues, like detections of simple constant-size objects like light switches and utility plugs, and some relative-depth clues like perspective lines. Aggregating such clues over the whole image, and accounting for global context like floor and ceiling locations, might plausibly give an impression of absolute, global depth. Figure 37 depicts a high-level overview of a feed-forward network that uses several cost functions to enforce the depth-estimation feature representation at fine, intermediate, and coarse levels. Such a network is designed to minimize the effect of shortcut finding, but I expect that it cannot fully address the problem of adversarial fooling and related brittleness until the network is provided with feedback-derived hypothesis testing.

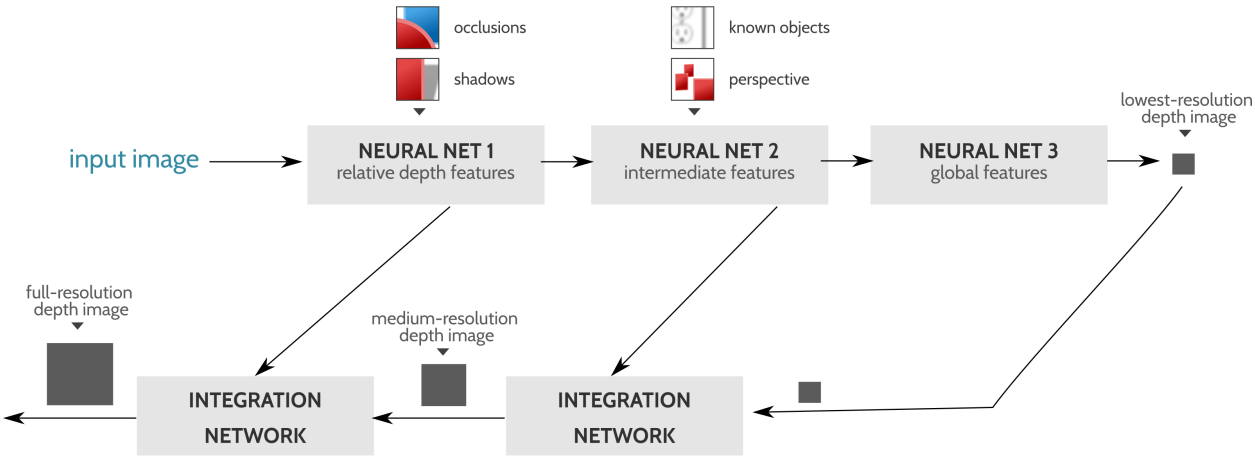


Figure 37: High-level depth-estimation network design

High-level network architecture is motivated by structural elements in images. Fine-scale features such as shadows and occlusions can both refine a depth map estimated by intermediate features and become building blocks of features of intermediate scale and complexity. Likewise, intermediate scale features play a role in refinement of depth maps estimated by scene-level features, and serve as building blocks of the scene-level features themselves. The lowest-resolution, medium-resolution, and full-resolution depth maps serve as semantically-meaningful ports at which loss can be computed independently, and at which the network can be connected to other networks.

5.4.2 Semantically meaningful ports

Another consideration that influences network structure is the desire to have semantically meaningful signals in several places within the network, rather than restricting interpretability of sig-

nals to the inputs and output of a long tube filled with indecipherable semantic juice. There are several related motivations for semantically meaningful ports. One motivation is in support of fortification via multiple cost functions as discussed; in order to have a cost function, there must be a meaningful quantity to evaluate. Another motivation is better qualitative assessment of how a network learns: the more windows into the network through which a system designer can observe measures of performance, the better the designer can build intuition for the network's performance. A third motivation for semantically meaningful ports is to enable separately-trained networks to be connected together at many interfaces. To progress toward the goal of densely-connected multimodal networks, it is a good and perhaps necessary engineering principle to achieve some modularity by training parts of the network to fill distinct functional roles. Good modularity requires good interfaces with meaningful abstractions. Finally and of most immediate interest, having ports in the network with defined semantics empowers neural networks with a propagator-like ability: the ability to combine signals from several origins. Having multiple networks use shared interfaces to negotiate the interpretation of co-occurring percepts in different sensory modalities evokes work by Coen [2006] on cross-modal clustering. As I describe in detail in Section 5.6.5, it is possible to introduce external signals into the neural network through its ports, to correct errors and improve performance. This capability holds additional promise as a way to implement feedback without encountering the gradient badness previously discussed, and is thus an exciting opportunity for future work.

5.4.3 Empirically successful foundation

An additional contributing factor to design decisions is the desire to use empirically-validated techniques in the design of the new network. Deep learning's most esteemed practitioners advise against excessive application of creativity in new designs[7] and my own experience corroborates that the advice is sound, albeit dissatisfying. My early attempts at building deep neural networks with unusual topologies drifted into the weeds of uncharted optimization problems without clear solutions of any kind. It was only by adopting conservative designs, that were able to benefit from transfer learning and from established methods of tuning hyperparameters, that I was able to experience more positive outcomes.

In this respect it is beneficial that the network depicted in Figure 37 has the hourglass structure with skip connections that characterizes many successful generative networks. For example, the semantic segmentation network of Long et al. [2015] and the depth estimation network of Kuznietsov et al. [2017] both have similar topology to the network structure depicted in Figure 37, namely, a directed acyclic graph (DAG) consisting of a main backbone with skip connections. The main feed-forward portion of my network is designed so that it can be initialized with pre-trained weights of VGG-19 (Simonyan and Zisserman [2014]) for faster training via transfer learning from VGG-19, which was trained for image classification.

5.5 Implementation

I considered many variations on the high-level architecture depicted in Figure 37. The end result is depicted in Figure 38. The only structural difference between the implemented network and the

[7] Advice given in a presentation by Andrew Ng entitled *AI is the New Electricity* at MIT on November 6, 2017. Similar advice is contained in the practicum chapter of Goodfellow et al. [2016].

high-level schematic depicted in Figure 37 motivated by the desiderata I identified is the addition of skip-forward connections. The skip-forward connections carry the input image, adjusted for scale, forward to the intermediate and final depth-estimation side-chains of the network. I found this addition to substantively increase the apparent sharpness of the depth maps. In the network, the signal passed from side-chain A to side-chain B and the signal passed from side-chain B to side-chain C represent the internal signals with interpretable semantics. At every pixel location of side-chain A's $\frac{1}{16}$ -scale output and side-chain B's $\frac{1}{8}$ -scale outout, the 64 vector components of each of these signals linearly combine to form reciprocal-depth estimates.

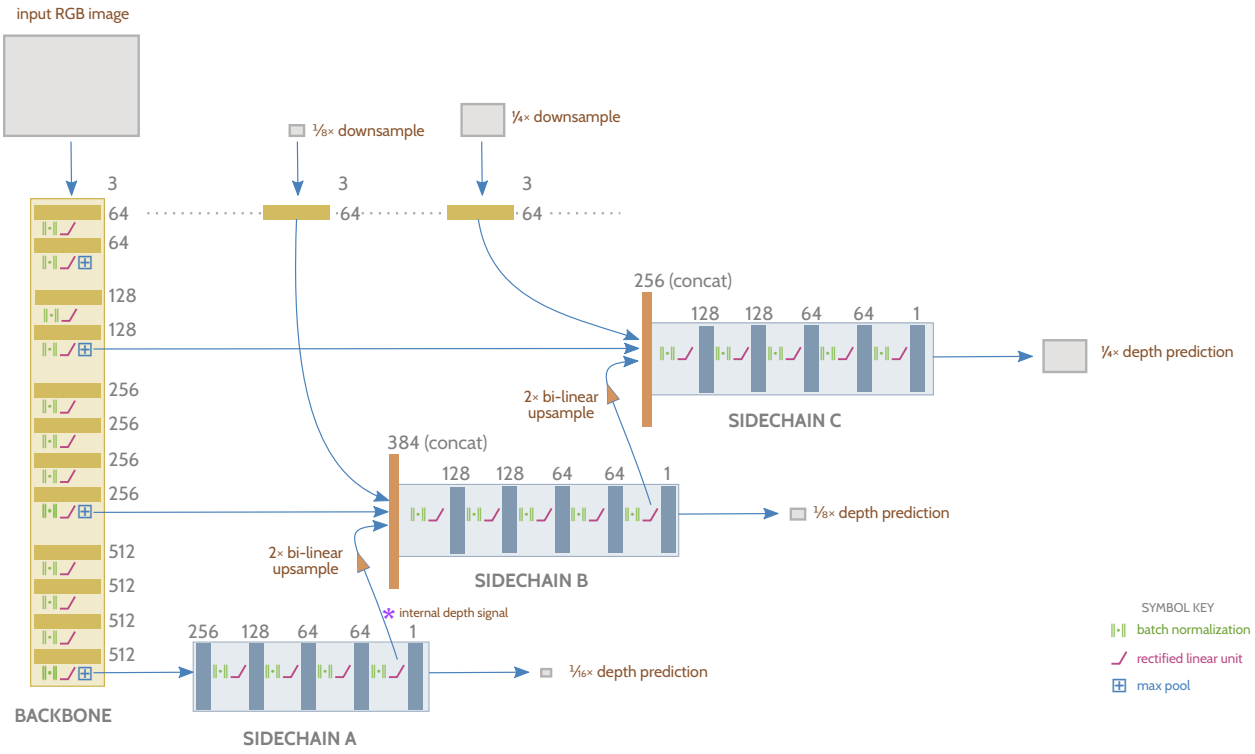


Figure 38: Neural network for depth estimation

The backbone of the network, shown in gold, is initialized to the corresponding weights of VGG-19. The side-chains are randomly initialized. The convolutional layers in skip-forward connections are each initialized to the first-layer weights of VGG-19. All convolution kernels are 3-by-3 except the side-chain outputs which are 1-by-1. The purple asterisk above side-chain A marks the point of entry for alternatively-sourced signals. The internal signals linking side-chains A and B, and side-chains B and C have straightforward interpretations because their components combine linearly to produce depth estimates at different scales.

The most successful training strategy among those that I tried was to fix the weights originating from VGG-19 and aggressively train the randomly initialized weights, and then finalize the network by training all weights simultaneously and less aggressively. I present the details of this training procedure in Section 5.6. The most successful cost function among those I applied was the mean-squared error (MSE) of the reciprocal depth function:

$$\mathcal{L}(R, R^*) = \frac{1}{n}\sum_{i}^{n}(R_i - R_i^*)^2 \tag{16}$$

In this MSE loss, n is the number of valid depth values in the ground-truth depth map R^*, as specified by a mask bitmap. R and R^* are defined as the element-wise reciprocals of the estimated and ground-truth depth maps Y and Y^*, respectively:

$$R_i = \frac{1}{Y_i} \quad , \quad R_i^* = \frac{1}{Y_i^*} \tag{17}$$

The choice to have the network output reciprocal depth rather than depth follows from the geometric properties of measurements made by the Kinect sensor. The sensor works by projecting a constant, non-uniform pattern of near-infrared dots on the scene using an IR laser and a diffraction grating. It determines parallax by measuring the projected pattern with a dedicated IR camera and then correlating the measured projected pattern with a template. Depth is proportional to the reciprocal of the parallax measurement.

Parallax is the Kinect sensor's intrinsic type of measurement: the IR camera that measures the projected pattern has a grid of equally-spaced sensors, and parallax is directly proportional to pixel displacements on this sensing grid. Whereas it is common practice to have networks learn to represent the log of depth (Eigen et al. [2014]) or to represent the depth values directly (Eigen and Fergus [2014]), reciprocal depth is, in principle, better suited to the capabilities of the sensor. Reciprocal depth is asymptotic as depth increases whereas both log depth and direct depth are unbounded. The reciprocal, *native* loss function therefore does not unfairly punish errors in distant objects that cannot be accurately represented in the ground-truth data. Of course, the reciprocal depth prediction and associated family of loss functions might be less appropriate for a sensor that operates via a different fundamental principle than parallax, such as time-of-flight LiDAR.

5.6 Experiments

In this section I describe the infrastructure, data, and other details of the training process. I present results of training the network with two different ways of handling an issue arising from scale-dependent features in the training data. I demonstrate how the semantically-meaningful ports of the network enable the network trained to estimate depth maps from images to also increase the resolution of existing depth maps, using images and low-resolution depth maps as input.

5.6.1 Training infrastructure

All training was carried out by a single GNU/Linux node equipped with dual six-core Xeon X5690 processors clocked at 3.47 GHz, 96 GB of system memory and 4 GTX 1080 GPUs with 8 GB of GPU RAM each. TensorFlow (Abadi et al. [2015]) version 0.12.0-rc0 provided all neural network

and optimization routines. GPU memory per card revealed itself initially as a bottleneck when training large networks or using large batch size, so I created a system of abstractions enabling more flexibility in splitting up networks and training jobs across several GPUs. The abstractions were of crucial importance to train certain models on the hardware, but they needed to accommodate interactions between models of parallelism and the details of neural network techniques such as batch normalization (Ioffe and Szegedy [2015]), and GPU architectural minutiae such as DMA channels. Section A.5 contains in-depth treatment of selected technical issues addressed by my abstractions.

File-system read performance created another bottleneck. The training throughput of the 4 GPUs, when properly configured, exceeded the read performance of the file-system more than 10 fold, and though a variety of database solutions to such problems exist it was more expedient to build a purpose-specific data pipeline than to deploy an existing database. The data pipeline consists of a 22 GB in-memory ring buffer into which a tunable number of dedicated threads read training examples from disk in order to maximize file-system throughput. The oldest training example in the buffer is overwritten whenever new data is ready. Batches of training examples are constructed by randomly selecting examples from the approximately 10,000 640x480-pixel 24-bit RGB images and corresponding 32-bit floating-point depth maps that fill the ring buffer. In addition to the regularization induced by this batch-shuffling process, data augmentation in the form of randomly-applied cropping, horizontal flipping, scaling, rotation, brightness shift, color shift, and contrast adjustment additionally contribute to diminishing the effect of the relatively slow turnover rate of training examples in the ring buffer. Every time the system draws a sample from the ring buffer, the system applies the randomized data-augmentation procedures to it, resulting in effective size amplification of the pool of examples present in the buffer.

The color, contrast, and brightness data-augmentation procedures are implemented by TensorFlow, whereas the cropping, rotation, and scaling are performed first by an external training harness in order to make use of the spare CPU time that would otherwise be spent idle during disk-read and GPU operations. The augmented training data are accumulated in several FIFO queues that are opportunistically filled. The data pipeline architecture is implemented as a collection of modular components that can be instantiated as separate processes that communicate via local sockets so that the pipeline start-up delay need only be incurred once during neural network debugging. Alternatively, all pipeline components can exist as objects within the same process that communicate via shared memory, to lower communication overhead during long training runs. All data-pipeline components are implemented in the Python language, and CPU-intensive image transformation is achieved with OpenCV (Itseez [2015]) version 3.1.0.

5.6.2 Training data

To train the neural network I used the raw portion of the NYU Kinect Dataset Version 2 (NYU Depth) (Silberman et al. [2012]).[8] NYU Depth contains 407,024 image-depth pairs from 464 scenes of cluttered indoor environments. I divided the scenes by the authors' convention into train and test subsets and trained on the 249 scenes in the authors' train subset. To test the networks, I used the labeled version of the dataset, which consists of 654 image-depth pairs drawn from the 215 test scenes. These 654 depth maps are infilled so that there are no invalid regions, which

[8] A Free version of the NYU Depth Toolkit, which I translated from a proprietary platform to SciPy (Jones et al. [2001–]) and OpenCV (Itseez [2015]), will be made available to download from the NYU-Depth website.

are common in the raw data due to the Kinect sensor's response to occlusions, certain types of lighting and specular surfaces. Additionally, these curated images were chosen by the dataset designers not to contain sensor errors and misalignments that are not infrequent among the raw data.

5.6.3 Training particulars

I initialized the VGG backbone of the network shown in Figure 38 with the pre-trained weights of the VGG-19 network, pre-processed by the procedure described in Section A.4 in order to retrofit the network with batch normalization. Other weights were initialized to random Gaussian noise with $\mu = 0$ and $\sigma = 0.1$. Training used an L2 weight-decay regularization constant of 10^{-4}, a batch size of 128, and Adam optimization (Kingma and Ba [2014]) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The batch-augmentation parameter used in the multi-GPU batch-normalization hack discussed in Section A.3 was fixed at 0.5.

I experimented with many strategies and schedules for training the different parts of the network. The most valuable lesson from this experimentation was that there is clear benefit, in terms of convergence time and qualitative network performance, to training by applying the loss of Equation 16 simultaneously to the $\frac{1}{16}$, $\frac{1}{8}$, and $\frac{1}{4}$ -scale depth outputs that the network produces. The observed superiority of this approach corroborated the intuition of Section 5.4.2 that it is a beneficial engineering principle to build networks with semantically-meaningful ports.

I found that the training strategy that resulted in the fastest decrease in training and validation loss was to fix the weights that are initialized from the pre-trained VGG-19 network, and train the side-chain networks with an initial learning rate of 0.01. I multiplied the learning rate by a factor of 0.1 each time the network appeared to approach an asymptote in validation performance, stopping when the learning rate reached 10^{-5}, which happens to be the same final learning rate used to train VGG-19 according to Simonyan and Zisserman [2014]. I then trained the whole network at this final learning rate until I perceived no qualitative improvement in side-by-side comparison of validation output between snapshots of the network over time. This last qualitative fine-tuning step caused noticeable improvement in the network's output even though the change in measured validation error was insignificant. It is unclear whether different learning-rate schedules and training policies would produce better results; I settled on the protocol I described by trial and error over many training runs and I chose this particular method based on superior rate of improvement. Based on this experience I believe other methods of adjusting learning rate would likely work as well.

The inputs to the network during training were 224x224 image crops, generated by randomly selecting a square region of image with a side length between 320 and 480, applying data augmentation, and then re-sizing the image to 224. Data augmentation used random rotations of $\pm 20°$, random $\pm 5°$ shifts in hue, random contrast and brightness adjustment between 0.9 and 1.1, and a random decision to flip horizontally. All probability distributions for data augmentation were uniform. The image mean, computed over the whole training corpus, was subtracted from the input images. This mean was per-pixel, computed over the training data and their mirror images, and blurred to remove artifacts. I tried several strategies for adjusting depth-map values along with image scaling, discussed in Section 5.6.4. The cost function was MSE in the reciprocal depth, as discussed in Section 5.5.

5.6.4 Evaluation

In order to obtain a preliminary assessment of the system's performance I evaluated it using the error quantifiers used in prior depth-estimation work along with additional quantifiers. I emphasize that evaluation by metrics is not the end goal of my work; premature emphasis on numerical measures of performance invites a severe and often-overlooked hazard, as I argue in Section 1.3 and provide evidence for in Chapter 2. Bearing this caveat in mind, I measured the system's performance with the quantifiers Eigen et al. [2014] used in their work, adding to that list a measure of error in reciprocal distance that is equal to the loss used to train the highest-resolution output of the network, with regularization terms removed. The definitions of the error quantifiers are presented in Table 1. In the quantifier definitions in the table, N_p is the number of elements in a single depth map which is equal to the number of pixels in a (re-sampled) test image. The notation y_i denotes a single element of y. The depth estimated by the network is denoted by y with corresponding ground truth y^*. The test set is T.

Table 1: Definitions of Evaluation Error Quantifiers

Error quantifier	**Definition**
Threshold	average fraction of y_i such that $\max(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}) < T_{thr}$ for $1 \leq i \leq N_p$
Absolute relative difference	$\frac{1}{\lvert T\rvert} \sum_{y \in T} \lvert y - y^* \rvert / y$
Squared relative difference	$\frac{1}{\lvert T\rvert} \sum_{y \in T} \lVert y - y^* \rVert^2 / y$
RMSE	$\sqrt{\frac{1}{\lvert T\rvert} \sum_{y \in T} \lVert y - y^* \rVert^2}$
RMSE, log	$\sqrt{\frac{1}{\lvert T\rvert} \sum_{y \in T} \lVert \log y - \log y^* \rVert^2}$
RMSE, log, scale invariant	$\sqrt{\frac{1}{\lvert T\rvert} \sum_{y \in T} \lVert \log y - \log y^* + \alpha(y, y^*) \rVert^2}$ where $\alpha(y, y^*) \equiv \frac{1}{N_p} \sum_{i=1}^{N_p} (\log y_i^* - \log y_i)$
RMSE, reciprocal	$\sqrt{\frac{1}{\lvert T\rvert} \sum_{y \in T} \lVert \frac{1}{y} - \frac{1}{y^*} \rVert^2}$

Notational shorthand represents element-wise reciprocal as $\frac{1}{y}$ and element-wise natural log as $\log y$.

Table 2 shows the results of two trained networks compared with results that approximate the state of the art on this dataset, and to the value of the metrics using the data mean as the estimate. The network represented by the first column has the normal perspective-derived scale factor applied to depth-map values: scaling the training image by a factor of Z scales the depth values by $1/Z$. I noted when evaluating this network that the optimal scale factor, which minimized RMSE on the training data, was 1.14, when the scale factor due to the actual image scale transformation was 1.67. The disparity suggests that the network had difficulty learning the scale relationship

between image features and depth maps. A speculative cause of the difficulty is that max-pooling layers in the network, which impart some scale invariance to feature representations, prevented the network from making full use of the relationship between scale and depth.

When I removed the scale factor entirely during training, that is, when I re-scaled the image without applying any transformation to the corresponding values of the depth map, the measured optimal scale factor became 0.998. Performance on the RMSE metric defined in Table 1 worsened, but the depth maps remained qualitatively similar. Qualitatively better results appeared when I trained the network with a scale factor that was randomized so that the depth scale was decoupled from the scale of the image. The measured optimal scale factor was likewise close to 1, and the performance measures substantively the same as the performance measures for normal scale (in which the empirical optimal scale factor of 1.14, rather than the geometrically-motivated value of 1.67, was applied at test time). Investigating the cause of the better qualitative results with the less informative augmented training data is an opportunity for future work, but I speculate that it is because the training with unreliable scale more effectively removes the effect of scale-dependent features on learning, thus forcing the network to learn features that are good scale-invariant indicators of depth. Because max pooling does not interfere with the scale-invariant features, the performance of the network appears better. Figure 39 depicts the output of the normal-scale network and the decoupled-scale network on several images from the test data. Though both exhibit blurred edges that are characteristic of MSE loss, the decoupled-scale network achieves more realistic performance.

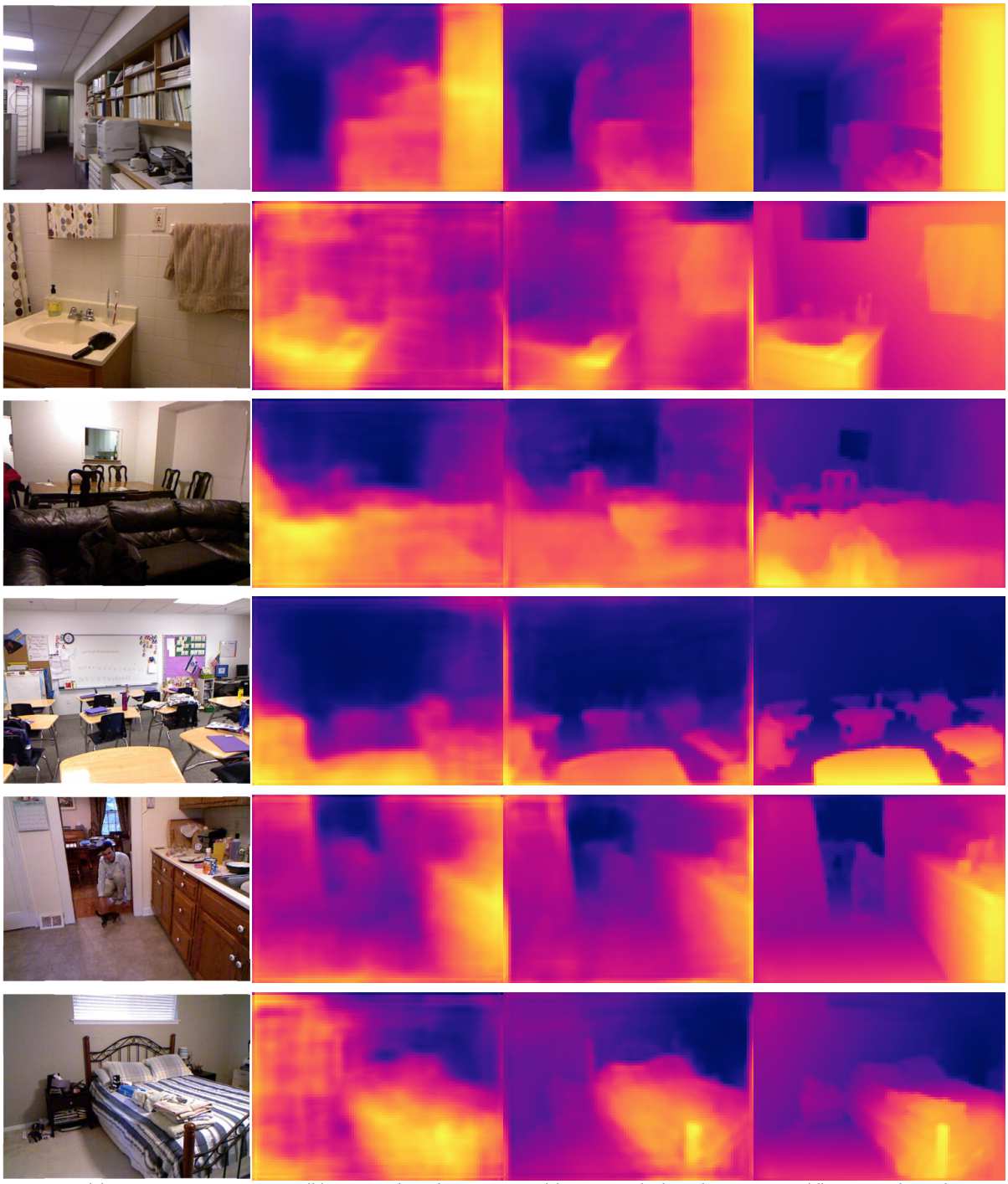
Table 2: Values of Selected Error Quantifiers

Error quantifier	Normal Scale	Decoupled Scale	E.F.	Mean
Threshold $T_{thr} = 1.25$	0.58	0.60	0.77	0.42
Threshold $T_{thr} = 1.25^2$	0.87	0.88	0.95	0.71
Threshold $T_{thr} = 1.25^3$	0.96	0.97	0.99	0.87
Absolute relative difference	0.24	0.22	0.16	0.41
Squared relative difference	0.20	0.21	0.12	0.58
RMSE	0.67	0.75	0.64	1.24
RMSE, log	0.28	0.27	0.21	0.43
RMSE, log, scale invariant	0.23	0.21	0.17	0.30
RMSE, reciprocal	0.16	0.13	–	–

E.F. refers to Eigen and Fergus [2014]. Mean results are from Eigen et al. [2014]. For the top 3 rows, higher values are better, and for the bottom rows, lower values are better. The decoupled scale results use a data pre-processing technique to reduce dependence on scale.

5.6.5 External signal introduction

A goal of the architectural choices discussed in Section 5.4.2 is to enable signals within the network to come from alternative sources. The principle of alternative sources for intermediate signals is a powerful idea taken from work on propagators (Sussman and Radul [2009]), that I identified in Section 4.4.5 as a key element to preserve from my efforts to use propagators in vision systems.



(a) Image (b) Normal scale (c) Decoupled scale (d) Ground truth

Figure 39: Outputs of two depth-estimation networks
Given the input image (a), the network trained without special attention to scale dependence (b) produces depth maps that exhibit lower RMSE than the depth maps produced by a network forced to ignore scale-dependent features (c), but this scale invariant network produces results that are qualitatively more similar to ground truth (d). All depth maps are independently normalized before coloring. The color map was designed by Kovesi [2015].

To take the first step toward realizing neural network architectures containing multiple, redundant information pathways characteristic of propagator systems, I configured the network depicted in Figure 38 to accept another input in addition to its main input, which is an image. The additional input is a specially-processed depth image at the lowest resolution estimated by the network of Figure 38, that is, a $\frac{1}{16}$ -scale depth map. The depth map is processed by another neural network, depicted in Figure 40, called an *ally* network to emphasize its cooperative role with the depth-estimation network. The ally network has been trained to estimate the internal signals of the depth-estimation network that are passed from side-chain A to side-chain B as depicted in Figure 38. Specifically, this internal signal is a 64-channel feature map with just $\frac{1}{256}$ the area of the input image. I trained the ally network shown in Figure 40 by running the same training data with data augmentation used to train the network architecture of Figure 38 using the best-performing weights learned by the decoupled-scale training method. The training objective for the ally network was to minimize MSE loss, and I used a learning rate schedule similar to that used to train the main network. Convergence to very low validation error happened quickly, which corroborates intuition that the pre-processing of raw depth to the internal signals of side-chain A is an easy function to approximate.

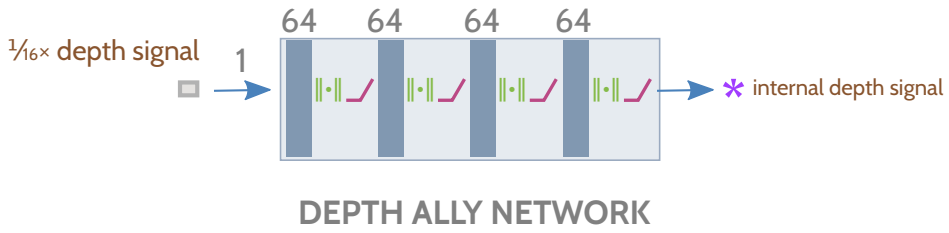


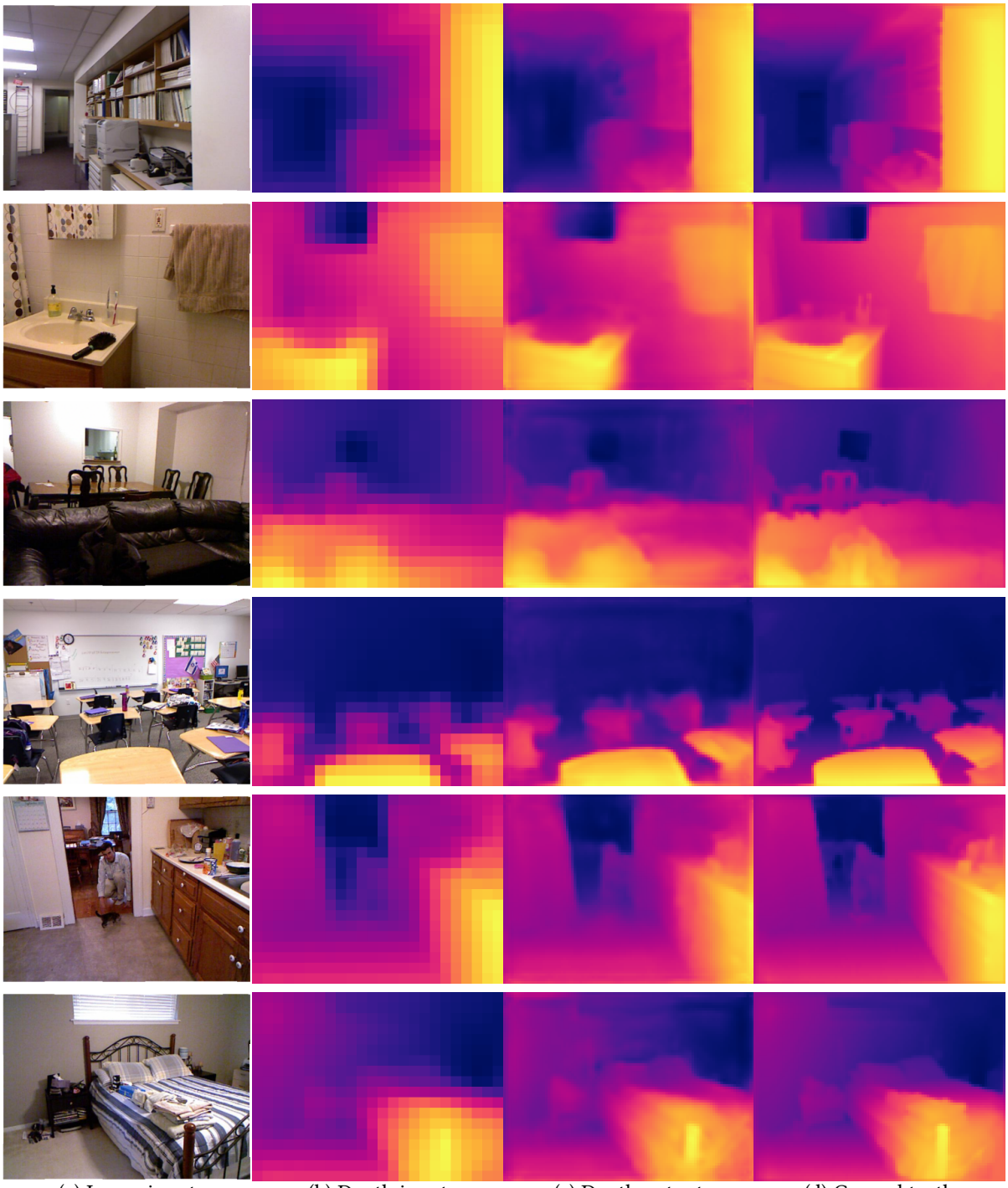
Figure 40: Ally network structure

The 5-layer ally network is randomly initialized and trained to estimate the internal signals marked by the purple asterisk in Figure 38 from a $\frac{1}{16}$ -scale depth map. The ally network can then be used to introduce external signals into the estimator network.

The ally network then provided a way to derive the lowest-resolution depth signal from an outside source, rather than from side-chain A of the estimator network. Figure 41 depicts example output of the network in one such configuration: the configuration where the $\frac{1}{16}$ -scale depth ground-truth value is supplied to the ally network and the resulting signal is supplied to the depth-estimation network at the point labeled by the purple asterisk in Figure 38 in place of the output of side-chain A. In this configuration, the two networks cooperate as a depth super-resolution network, using the images to increase the area resolution of the supplied depth map by a factor of 16.

5.7 Discussion

The neural network presented in this chapter is a step toward implementing several powerful ideas originating from the propagator architecture in a neural network framework. One powerful idea is that exposing semantically meaningful signals inside a network empowers existing machinery to combine in new ways to solve new problems. In the case of the propagator architecture, components can benefit from such reuse without needing to be redesigned, and in



(a) Image input (b) Depth input (c) Depth output (d) Ground truth

Figure 41: Signal-introduction depth up-sampling
The decoupled-scale network functions as a depth super-resolution network without retraining or modification. The images (a) and $\frac{1}{16}$ -scale down-sampled depth maps (b) are provided to the network, resulting in an up-sampled depth map (c) ($\frac{1}{4}$ -scale) that is very similar to ground truth (d).

the case of neural networks, without needing to be retrained via expensive supervised training. Another powerful idea is that computational flexibility arises from signal redundancy: when a network has several alternative ways to compute an internal signal under different conditions, it can opportunistically find new paths through the computation graph and hence novel solutions to problems. I combined these ideas to show that a network trained to estimate depth from images, when combined with an ally network that has learned to estimate one of the original network's internal signals, can solve a new problem of depth super-resolution without any retraining.

5.7.1 Extending ally networks

The ally network that I presented in this chapter is the simplest possible ally. It predicts the internal signal of another network that is known to linearly estimate depth, directly from the raw depth signal itself. To take the ally concept to the next level, the ally could learn to predict internal signals from a different source, for example, depth from optical flow, to achieve a structure-from-motion estimator. Similarly, it is not necessary to entirely train one network before training its allies. Training of ally networks can be symmetric: freezing a network and training an ally, then freezing the ally and training the original network, in turn. This is analogous to the way that adversarial networks are trained, except that the additional symmetry of the cooperative arrangement does not limit the number of participants to 2. I note the similarity between this method of training ally networks and work by Beal [2007] on casting learning as a communication-bootstrapping problem.

To take the ally concept yet a step further toward densely-connected cooperative network structures that can support robust visual intelligence, the process of combining allies' outputs can be automated. The network in the proof of concept that I described in this chapter contains configurable switches to choose its mode of operation: by estimating depth from an image alone, or from an image and a low-resolution depth map, or by making a pixel-wise binary decision as to whether the source of a signal derives from the depth estimator or the ally network.[9] A promising methodology to apply to achieving the desired automation is to make the switches continuously variable rather than binary, so that they produce an element-wise weighted combination of allies' outputs. Then, train other *routing networks* to control the switches. These routing networks could be trained independently of the ally nets. The routing network could, for example, accept all of the ally networks' outputs as its input, and use a softmax to control how the ally outputs are combined. Such a soft decision would enable a training strategy that alternates between mutually training ally networks in one phase, and then training routing networks in another phase. Once trained, the routing networks could then be used to achieve the best cooperation among ally networks under different conditions.

5.7.2 Applying immutable differentiable joints

Another promising direction for future development is to apply non-learned transformations to the signals exchanged between cooperating networks via their ports. Instead of being passed directly from one network to another, certain signals with known interpretation can be transformed programmatically between processing by the networks. A raw depth map, for example,

[9] The pixel-wise mode facilitates use of the network with a Kinect sensor, by allowing pixels for which the Kinect has valid depth data to come from the Kinect, others from the depth estimator.

has an interpretation as a set of 3-dimensional points. Those 3D points can be transformed, for example, rotated, translated, and re-projected onto the image plane to obtain a different depth map. Likewise, the 3D points can be used to re-project a color image to a new image plane. This transformation would give a downstream network a new perspective on the output of the upstream network. Importantly, as long as the transformation permits backpropagation of gradients through it, training the entire sequence comprised of the upstream network and the downstream network end-to-end with SGD is possible.

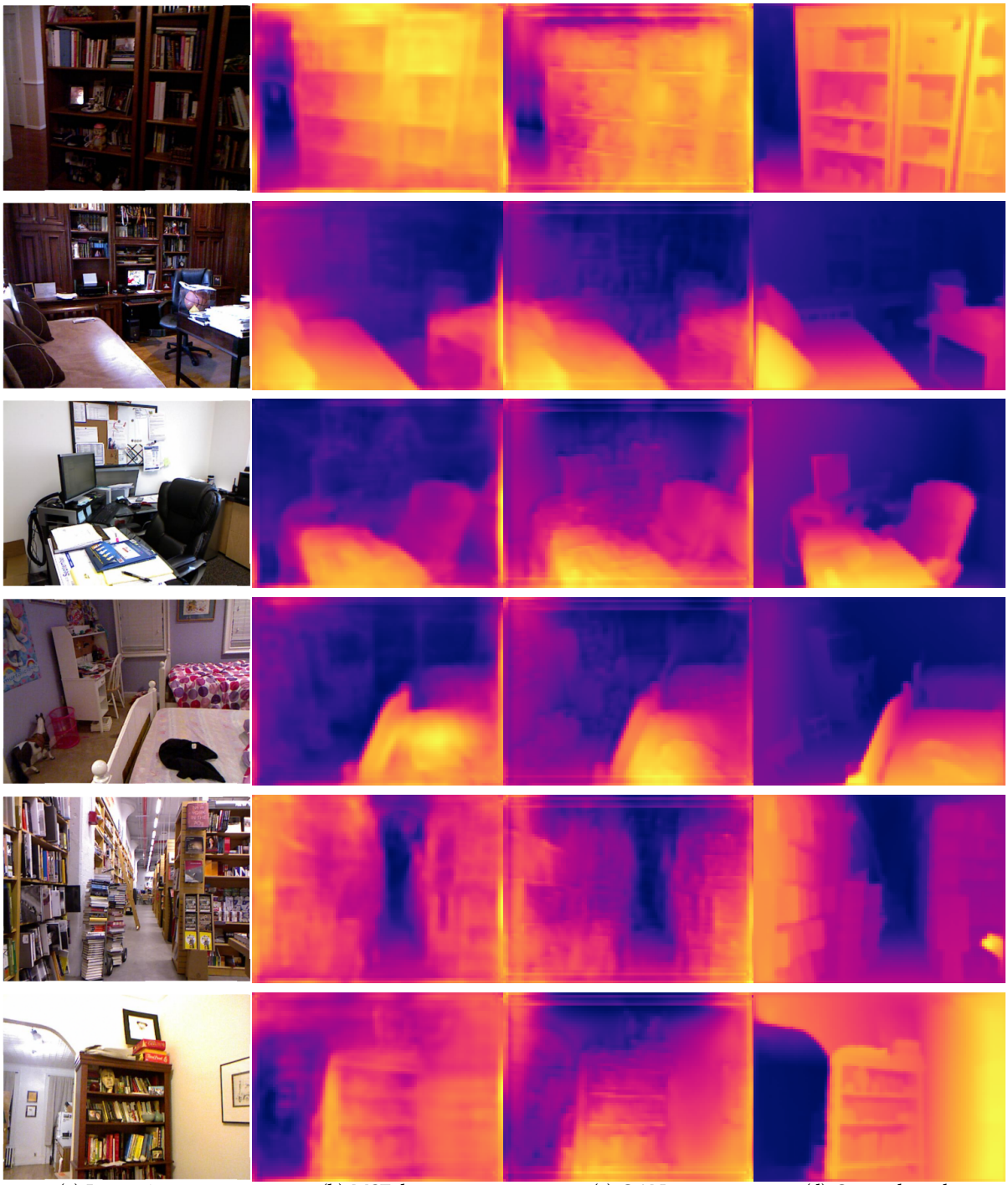
It is common to apply a fixed transformation to a network's output as part of a specialized loss function. With the advent of generative adversarial networks (GANs) (Goodfellow et al. [2014a]), it has become common to pit networks against each other in a contest in which the generator is honed by a complex evolving loss function represented by the adversary. I propose to interpose fixed transformations, derived from principled interpretation of internal signals that have specific meanings, between segments of neural networks in order to affect the dynamics of learning. For convenience I call the fixed transformations *immutable differentiable joints* (IDJs).

One way to apply IDJs in the depth-estimation problem that is the focus of this chapter is to place a re-projecting IDJ between the depth estimator and an adversarial discriminator, and to train the whole configuration as a GAN. I anticipate that this training strategy could improve upon preliminary results I attained using a GAN to train the depth estimator, depicted in Figure 42. The reason to anticipate improvement is that the re-projection of the image according to the processing of the estimated depth map by the IDJ could teach the adversary to recognize the difference between realistic image re-projections and unrealistic re-projections according to known rules of 3D transformation. In the GAN without re-projection by IDJ, which produced the results depicted in Figure 42, the depth map produced by the estimator and original image are left unmodified and presented to the adversary. As the results in Figure 42 show, the depth maps generated by the GAN have more fine detail and are less blurry than the depth maps generated by the MSE network. The next step in future work in this research direction is to apply the 3D-transformation IDJ between the generator and the adversary in the GAN.

5.8 Contributions

My main contributions in this chapter are as follows.

- I identified properties that make monocular depth estimation a good proving ground for neural-network-based alignment systems.
- I designed a neural network architecture for monocular depth estimation based on the principles that it should be resistant to adversarial attack and weak generalization of the type I investigated in Chapter 2, that it should have semantically meaningful ports, and that it should benefit to the maximum extent possible from transfer learning and established methods. The network achieves qualitatively good performance, and decent benchmark performance.
- I introduced a method of training depth-estimation networks to produce reciprocal depth rather than linear depth or log depth, and provided justification in terms of the operational specifics of the parallax-measurement device used to collect the training data.



(a) Image input (b) MSE-loss (c) GAN (d) Ground truth

Figure 42: GAN compared to MSE-trained network
The GAN in (c) passed both its depth estimate and the original image (a) to its adversary, which learned an adversarial cost function. The output of the MSE-trained network is shown for side-by-side comparison in (b), with ground-truth depth in (d).

- I observed that removing the correspondence between image scale and overall depth scale in the training data qualitatively improved depth-estimation results, and proposed a potential explanation for the observation in terms of limitations on the precision of scale correspondences that can be learned by a network with max-pooling layers.
- I introduced *ally networks*: independently-trained neural networks that learn to communicate via shared, semantically-interpretable feature representation. I used the ally-training method to enable a network trained for depth estimation to perform depth super-resolution without any retraining.

6 Summary of Contributions

To motivate this work I asked how we can account for the properties of natural vision that make it robust. I identified the properties that seem best to characterize this robustness: vision can perform many tasks consistently under highly variable conditions, it can adapt to new tasks, seemingly with with little preparation, and it appears to give us, its wielders, a rich compositional understanding of what it perceives.

I made observations about natural constraints that must have influenced vision's development. Reconstructing world state from images is fundamentally underdetermined, and further complicated by practical matters. Fortunately, though, we learn to see at the same time that we learn to interpret other senses, and so we can exploit rich correspondence and interdependence among the many ways we sense our surroundings. The correspondence elucidates strong constraints and overwhelming regularity that can be brought to bear on resolving the inherent ambiguity of seeing. Finally, there has always been strong selectional pressure on vision processes to be fast and directed toward survival goals.

The observations led me to characterize vision in terms of its computational imperative: in order to perform under the constraints imposed upon it by nature, and exploit opportunities provided to it, vision must opportunistically use multimodal information in order to resolve inherent ambiguity, and it must be able to prioritize goals.

The characterization of vision in terms of its computational imperative led to the fundamental question that motivated my work: can we completely account for vision's robust properties by way of processes that perform ubiquitous alignment among the elements of computation that work together to achieve multimodal perception? By alignment, I mean a process by which computational units reconcile their local state with that of their neighbors across shared interfaces. By requiring that this process be multimodal, I suggest that vision gains its robust qualities by recruiting the perceptual machinery of other senses. By requiring pervasive alignment, I reject fixed top-down or bottom-up data pathways in favor of opportunistic paths through a computation graph, driven by the needs of local units of computation to fill gaps in their local state. I label the fundamental motivating question the **alignment hypothesis** and call the type of models characterized by it visual alignment models.

High-performing computer vision models of today are not alignment models. Superficially, this would seem to detract from the line of research inspired by the alignment hypothesis. Observed more carefully, though, even the highest-performing models available suffer from certain conspicuous failures, despite their ability to match and exceed human visual ability by some measures. What makes the failures conspicuous is that they are inexplicable both by the computer-vision systems and by our own understanding of what they see. Furthermore, adversarial images and training examples that cause such errors can be reliably generated even for today's best performing systems. I argued that the existence of these problems rules out the possibility that the systems possess robust visual intelligence.

I investigated the nature of brittle failures of convolutional nets in experiments, detailed in Chapter 2, in which I sought to isolate the minimum sets of features contained in natural images that facilitate their classification by a neural network. I conducted an experiment to determine whether those minimal images were easily recognizable by people, and found that whether or not they were depended surprisingly on the *object class* represented by the image. I supposed that the problem used to measure performance, 1000-way image classification in the case of the

network under consideration, caused it to learn a classification strategy that led to good measured generalization without learning uniformly robust feature representations over all classes. I found evidence for the supposed strategy in the relative magnitude of signals at the highest level in the neural network.

I implemented an alignment-based vision system using a propagator mechanism. The system, described in Chapter 4, can track pedestrians and use their measured heights and locations to determine the location of the ground plane and the locations of occluding objects, within bounds. The system allows information to propagate between objects that interact: information about a known pedestrian height could affect the parameters of the ground plane, which in turn helps to estimate the positions and heights of other pedestrians. A review of the effort to build the system revealed insights, in Sections 4.4 and 4.4.5, into what to modify in the propagator architecture I used, to support its application in low-level alignment.

The insights about the propagator architecture led me to implement a neural network system for monocular depth estimation, with emphasis on realizing propagator-like capability in this network. In those efforts, detailed in Chapter 5, I developed methods to fortify against adversarial examples, and to ensure that the network contained ports with interpretable signals. The interpretable signals enabled a network trained for depth estimation to perform depth super-resolution without retraining, by sharing information with an ally network via shared signals communicated via the port. I anticipated future directions of this work to scale it to more challenging problems.

I see propagation as a metaphor for the scientific discovery process. Progress can be slow in the first steps toward solving a difficult problem. As more pieces of the solution fall into place, the pace of discovery quickens. A densely-connected propagation graph encourages communication and collaboration. I hope that as more people become interested in understanding the principles of human visual intelligence, we can communicate eagerly, sharing partial solutions to accelerate the pace of discovery of something that is beyond the grasp of our individual capabilities.

A Appendix: Neural Network Methods

Some of my work involved deep learning with neural networks. This technology has grown explosively in recent years, and although in many cases the results of applying deep learning to perceptual tasks in AI speak for themselves, the process itself of applying deep learning successfully is often arcane. This process involves composing vast networks from an increasingly diverse set of components, and choosing values for an ever-growing set of hyperparameters, whose interactions are difficult to predict, and sometimes even difficult to quantify due to the prohibitive combinatorics of their measurement. Making matters more complicated, the logistics of managing large datasets and the details of efficient distributed computation are unavoidable in any endeavor to achieve results on par with the state of the art in deep learning. Many resources have emerged to introduce the theory and practice of deep learning to people with varying levels of experience in machine learning. I found that the shortest and most essential reading list consists of **Deep Learning** by Goodfellow et al. [2016] and the UFLDL tutorials (Ng et al. [2013]). Additionally the TensorFlow software system (Abadi et al. [2015]) was indispensable to prototyping neural networks. TensorFlow's development is supported by a large team of industry experts, and I found it to provide superior usability, performance, and features to other frameworks, such as pioneering research framework Caffe (Jia et al. [2014]), that I used in earlier work described in Chapter 2.

A.1 Overview

In this appendix I share a more detailed presentation of techniques that I developed in the course of my work with neural networks than would be possible without detracting from the structure of Chapter 5. These techniques were hard-earned, and I hope that by sharing them I can help others confronted with the daunting challenge of embarking on a deep-learning research endeavor. This presentation includes techniques that were indispensable to the work described in Chapter 5 such as the Batch Normalization Retrofit procedure I present in Section A.4, and techniques which I developed but ended up setting aside, such as the method for bootstrapping fully-connected layers that I present in Section A.2.

This appendix is a loosely organized collection of mostly independent subsections related to some practicalities of designing deep learning systems. The topics covered range from software design patterns that I found useful for prototyping deep learning systems, to essential practical considerations required for efficient implementation of promising techniques from the literature. I also present some techniques that I found helpful in speeding up or otherwise enhancing training of certain types of networks.

A.2 Bootstrapping fully-connected layers from convolutional layers

Convolutional networks offer certain benefits over their less-constrained fully-connected (FC) counterparts. When a network operates on data in which spatially adjacent samples exhibit much greater correlation than distant samples, convolutional models naturally fit the data distribution better because they waste no effort searching for patterns that exceed the spatial extent of the convolution kernel. Additionally, because convolutional models reuse the same weights at many spatial locations, they have effectively many more training examples per network weight than FC

layers have. As a result of these differences between convolutional and FC layers, convolutional layers have lower feedforward computational cost, compact representation, and tend to train faster than FC layers because they can make more efficient use of each training example.

It is common practice to convert FC layers to convolutional layers after training, so that a network with FC layers can be used, for example, for semantic segmentation procedures that require a class label for every pixel in an image. The procedure of converting fully connected layers to convolutional layers was first described by Long et al. [2015]. I am not aware of any published work on the inverse of this operation, however, in which a convolutional network is converted to an FC network. Why would converting a convolutional layer to an FC layer be useful? The intuition is as follows: suppose that we would like a layer in a neural network to learn a function that we suspect is weakly local. That is, the output at a given location depends often on the input in a neighborhood around that location, but sometimes on distant inputs as well. A convolutional network can rapidly and efficiently learn the local dependencies that constitute an approximation of this function, but such a network is incapable of learning the dependencies that exceed the reach of the convolution kernel. An FC network, however, has to do a lot of work re-learning the local dependencies at many locations as it cannot benefit from weight sharing the way the convolutional network can. Therefore, what if we could treat the network as convolutional until it has learned the local dependencies, and then switch to treating it as fully connected so that it can learn the non-local dependencies?

In my work with depth estimation I experimented with network architectures in which the final stages of the network are FC and all preceding stages are convolutional. Such network organizations are common in the literature on both discriminitive and generative networks, but in the generative case the fully connected layers have image-like outputs with a lot of locality, and so they suffer from the inefficiency I described. In the end, I found that fully convolutional networks performed better than networks with a final FC layer, so I did not end up using the conversion procedure described in this section, but I still expect this type of conversion to be useful in some contexts. For example, in indoor depth-generating networks, global scene attributes such as floor and ceiling locations should create a strong bias for the depths of other objects in a room. In my initial experiments with convolutional-FC networks, training achieved little qualitative progress in several days on the 4-GPU system I used for training, whereas just a day of training a fully convolutional network produced markedly better results. Converting the trained convolutional layers to FC layers using the procedure described in this section produced marginally better results than the fully-convolutional system after only several more days of training. I ended up not using it only because the marginal benefits attained did not outweigh the additional computational cost of the FC layers.

I present here a procedure to convert a convolutional layer to an FC layer. I provide self-contained source code for doing the conversion along with an illustration generated directly from the included source code.

The key to understanding the implementation is that convolutional layers are special cases of fully-connected layers, in which the weights of non-local connections are zero, and the weights of local connections are replicated across many network units. The process of converting a convolution kernel to an FC matrix is illustrated in Figure 43, which is generated by supplying an example 5x5x1x1 kernel to the source code provided in Listing 2. The resulting matrix, when multiplied by the row vector obtained by flattening the image in row-major form, produces a result equivalent to convolving the 4x6 image with the supplied kernel. Because many of the

elements of the converted FC matrix are equal to zero, SGD would leave them unchanged and the resulting FC layer could not learn relationships between spatially-distant signals. To account for this and allow the FC layer to learn relationships between spatially-distant signals, the final step in the procedure is to add small-magnitude zero-mean noise to the converted FC matrix before continuing training in FC mode.

```
import numpy as np
def compose_blocks(blocks,grid_dimension):
    block_shape = blocks[0].shape
    pad = np.zeros_like(blocks[0])
    if grid_dimension > len(blocks)/2+1:
        padding = [pad]*(grid_dimension-len(blocks)/2+1)
    else:
        padding = []
    padded = padding+blocks+padding
    rows = []
    for i in range(grid_dimension):
        idx = len(padded)/2-i
        assert idx>=0,idx
        rows.append(np.concatenate(padded[idx:idx+grid_dimension],1))
    return np.concatenate(rows,0)
def row_to_block(kern_row,conv_row_len):
    row_blocks = [np.transpose(kern_row[i,:,:])\
                  for i in range(kern_row.shape[0])]
    return compose_blocks(row_blocks,conv_row_len)
def conv_to_fc(K,rows,cols,additive_noise_factor=0.0):
    """
    K: conv kernel to turn into a fc matrix
    rows,cols: dimensions of the input that would be convolved with K.
    additive_noise_factor:
        if non-zero, multiply the std dev of the convolution matrix
        by additive_noise_factor, generate zero-mean Gaussian noise
        with the resulting std dev, and add it to the convolution
        matrix. this creates a small disturbance that makes learning
        possible.

    output: a matrix W such that reshape(matmul(flatten(X),W),rows,
        cols) == conv2D(X,K)

    K must be of shape [height,width,ins,outs] The mode of the
    convolution is assumed to be SAME, so that a zero border is added
    to x and the output of the convolution is also size rows,col.
    Convolution is assumed to have spatial stride of 1.
    The kernel width and height must be odd.
    """
    kh,kw,kin,kout = K.shape
    assert kh>0 and kw>0,(kw,kh)
    assert kh%2==1 and kw%2==1,(kw,kh)
    hpad = kh/2
    wpad = kw/2
    superblocks = map(lambda row:row_to_block(row,cols),
                      [K[i,:,:,:] for i in range(kh)])
    mat_conv = np.transpose(compose_blocks(superblocks,rows))
    if additive_noise_factor:
        var = np.var(K)
        gen_scale = additive_noise_factor*np.sqrt(var)
        additive_noise = np.random.normal(size=mat_conv.shape,
                                  scale=gen_scale).astype(np.float32)
        return mat_conv+additive_noise
    else:
        return mat_conv
```

Listing 2: Complete code to convert convolution kernel weights to FC matrix

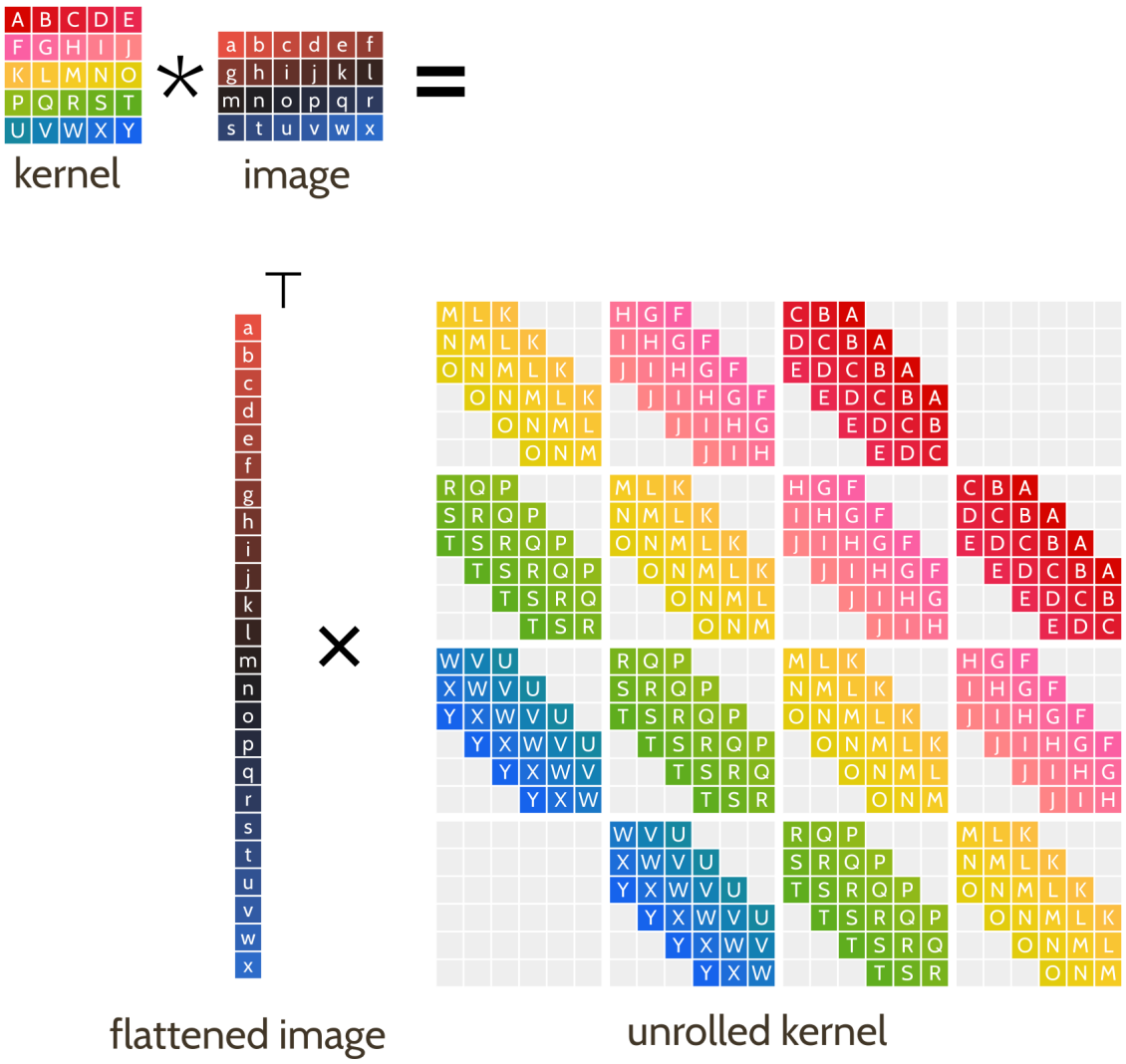


Figure 43: Conversion of a convolutional layer to an FC layer

Example output of the provided code. The function `conv_to_fc()` applied to the supplied 5x5 kernel with an input/output image size of 4x6 produces the fully-connected matrix shown. Gray areas have zero value. In order to allow gradient-descent learning to learn non-local dependencies, small valued noise must be added to the FC matrix before continuing to learn. The color maps used here were designed by Kovesi [2015].

The process of converting convolutional layers to FC layers could prove useful in refining generative networks that output a signal that is expected to have some degree of spatial locality constraint, but that may have non-local interactions as well. An important caveat is that, in such FC layers as these with many-to-many mappings, the computational cost is significantly increased from that of a convolutional layer with the same shape, and deep networks composed of full-resolution fully connected layers could remain infeasible for a long time yet. When special-purpose hardware for deep learning finally becomes mainstream, I expect that the provided method of hybrid convolutional-FC training will be useful to someone.

A.3 Batch normalization on multi-GPU systems

Batch normalization improves optimization performance in deep networks by reducing internal covariate shift (Ioffe and Szegedy [2015]). Additionally, batch normalization provides a source of regularization. Batch normalization relies on local statistics of mini-batches of data, and its performance is sensitive to the mini-batch size. Unfortunately, batch size is also a crucial performance-determining factor of GPU hardware, in which GPU memory is limited and transfers to and from GPU memory constitute major performance bottlenecks. I developed modifications to the batch normalization algorithm that reduce the sensitivity of batch normalization to batch size, so that limitations imposed by the hardware have reduced effects on the performance of batch normalization.

A.3.1 Overview of batch normalization

This section provides a high-level overview of the mechanics of batch normalization. Refer to the original paper by Ioffe and Szegedy [2015] for detailed motivation and analysis of the technique.

As deep networks learn, the distributions of the internal signals change. This change can be problematic when it causes nonlinearities within the network to saturate, driving their derivatives toward zero. The effect of the small-magnitude derivatives is amplified by the chain rule, leading to a vanishing gradient and poor optimization performance. Poor optimization performance also results when the internal signals have large variance, resulting in instability. Batch normalization addresses these problems by approximately normalizing the internal signals of the network at every training step. Because it would be computationally expensive to normalize the internal signals according to the statistics of the entire dataset at every training step, batch normalization uses statistics measured within the same mini-batch that is used in stochastic gradient descent.

In a batch-normalized network layer, batched inputs $\boldsymbol{u}$ are multiplied by a weight matrix $\boldsymbol{W}$ to produce a batch of D - dimensional intermediate signals $\boldsymbol{x}$, and then a batch mean $\overline{\boldsymbol{x}}$ and batch standard deviation $\boldsymbol{s}$ are computed with respect to the elements of this batch of intermediate signals. The process then computes the normalized intermediate signal $\widehat{\boldsymbol{x}}$ by subtracting the batch mean and dividing by the batch standard deviation (with numerical stabilization):

$$\widehat{x_d} = \frac{x_d - \overline{x_d}}{s_d + \epsilon} \quad \text{for} \quad d = 1...D \tag{18}$$

The signal $\widehat{\boldsymbol{x}}$ is then multiplied element-wise by a learned scale parameter $\boldsymbol{\gamma}$ and added to a learned mean-shift parameter $\boldsymbol{\beta}$, and finally passed through a nonlinearity such as $ReLU(\cdot)$ to produce the output, $\boldsymbol{y}$.

$$\boldsymbol{y} = ReLU(BN_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{x})) \quad \text{where} \quad BN_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{x}) \equiv \boldsymbol{\gamma} \odot \widehat{\boldsymbol{x}} + \boldsymbol{\beta} \tag{19}$$

The scale and mean-shift parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are trained via backpropagation alongside the weights. There are two additional parameters updated outside of the optimizer: the population mean $\boldsymbol{\mu}$ and the population variance $\boldsymbol{\sigma^2}$. These are used in inference mode in place of the batch

statistics, because in inference mode there may not be a batch of examples to compute statistics from, and because it is desirable for the inference-mode output of the network for a given input example to depend only on that example. Because parameters are fixed in inference mode, the batch normalization becomes just an affine transformation of the input. Rather than compute the true population statistics after training of the network is finished, the batch normalization method approximates these statistics by averaging them over recent training steps using a decay constant, $0 < r < 1$.

$$\boldsymbol{\mu}_{t+1} = r\boldsymbol{\mu}_t + (1 - r)\overline{\boldsymbol{x}}_t \tag{20}$$

$$\boldsymbol{\sigma^2}_{t+1} = r\boldsymbol{\sigma^2}_t + (1 - r)\frac{M + 1}{M}\boldsymbol{s^2}_t \tag{21}$$

where $\boldsymbol{s^2}$ is the batch variance, M is the batch size, and t is the training step. Initialization of the population statistics variables is unimportant because the population statistics do not affect training of other variables, and the decay rate must be short enough to forget obsolete batch statistics as the network learns and changes.

A.3.2 Modifications to the batch normalization algorithm

GPU memory size and the mechanics of TensorFlow models together impose severe constraints on batch size for complex models. I found that for one adversarial network architecture with a maximum depth of 38 layers, resource constraints precluded batch sizes of more than 12 examples per 8GB GPU module. Although SGD scales easily on a multi-GPU system via gradient aggregation and averaging, batch normalization does not scale in an analogous way because synchronizing batch statistics between GPUs would become a bottleneck. A batch size of 12 is at least an order of magnitude below what is required to achieve learning stability in many cases, so I modified the batch normalization algorithm to be more stable with a smaller batch size.

The motivation for the approach I used to address the GPU memory issue was to simulate batch augmentation with virtual examples, drawn from the global population, without having to allocate memory for those additional examples. Specifically, I sought to simulate the effects of adding $M - m$ virtual examples to the small batch of size m, in order to have an effective batch size of M, by estimating the statistics of a batch of size M from the true statistics of the batch of size m and the approximate population statistics. If it were the case that the population statistics variables perfectly represented the true population statistics, there would be no need for batch statistics at all. Because the system is learning while the population statistics are being aggregated, though, the variables do not contain the true population statistics at the current training step. The batch examples are drawn from the true data distribution at the current training step, but because the batch is so small, the statistics tend to be noisy. The key idea is to balance these two imperfections.

Suppose there were no nonlinearity. Then by the definition of batch normalization, the $\boldsymbol{\beta}$ parameter at layer $\ell - 1$ could always predict the mean at layer ℓ by linear transformation with the weight matrix: $\boldsymbol{\mu}_\ell = \boldsymbol{\beta}_{\ell-1}\boldsymbol{W}_\ell$. Unfortunately, without knowing the shape of the distribution

of the signal prior to application of the nonlinearity at layer $\ell-1$, this convenient linear method no longer works in the presence of nonlinearity. Complicating matters further, there is no analogous way to predict the variance at ℓ given only mean and variance at $\ell - 1$, regardless of whether the layers contain nonlinearity functions.

Because there seems to be no flawless way to simulate the desired virtual batch of size M, I engineered a workaround by adding an additional hyperparameter $q_{batch-augmentation}$ (q for short, within this section) that controls mixing of the batch statistics $\overline{\boldsymbol{x}}$ and $\boldsymbol{s}$ with the running average parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. For $0 \leq q \leq 1$:

$$\boldsymbol{\mu_q} = q\overline{\boldsymbol{x}} + (1 - q)\boldsymbol{\mu} \tag{22}$$

$$\boldsymbol{\sigma_q} = q\boldsymbol{s} + (1 - q)\boldsymbol{\sigma} \tag{23}$$

Then the parameters $\boldsymbol{\mu_q}$ and $\boldsymbol{\sigma_q}$ are used respectively in place of $\overline{\boldsymbol{x}}$ and $\boldsymbol{s}$ in the batch normalization step described by Equation 18. In practice, the hyperparameter q seems to have a wide range of values that permit stable optimization. For values of q that are too high, optimization is unstable because of the small effective batch size. If q is set too low, and at high learning rates, optimization also becomes unstable, possibly because of the time delay introduced.

A.4 Improving transfer learning with batch normalization retrofit

Deep feature learning from scratch requires large datasets and substantial computational resources. If feature representations learned in the course of solving one type of problem are applicable to another related problem, then it substantially improves learning efficiency to bootstrap the model intended to solve the new problem from the pre-trained network that does a good job of solving the old problem. TensorFlow (Abadi et al. [2015]) and other deep learning frameworks provide platforms for exchanging both model definitions and model parameters, making such transfer learning from existing models even more appealing due to ease of implementation. I used a transfer learning strategy to initialize my depth-map generating network from VGG19 (Simonyan and Zisserman [2014]), a network trained on ImageNet (Russakovsky et al. [2015a]) to classify images.

VGG19 was developed before the advent of batch normalization (Ioffe and Szegedy [2015]), a powerful technique that simultaneously provides regularization and reduces the problem of gradient vanishing and explosion[10]. In the course of my work on bootstrapping depth-map generating networks from VGG19, I found evidence that the gradient used to train the network was sometimes vanishing or exploding in magnitude, preventing optimization from making progress. Table 3 exhibits the source of the problem: the large magnitude of both mean and variance of signals deep within the VGG19 network. These signals are amplified by the chain rule to produce derivatives that are numerically unwieldy, causing poor optimization performance.

To resolve the numerical problems and to realize the regularization benefits of batch normalization, I developed a method to retroactively add batch normalization layers to a pre-trained

[10]Refer to Section A.3.1 for a high-level overview of batch normalization.

Table 3: Measured means and variances of VGG19 internal signals before application of the BNR method

Signal Name	Mean	Variance
conv1-1	0.496474	4509.737572
conv1-2	65.376059	35687.675124
conv2-1	18.005497	103246.684329
conv2-2	-63.906908	185671.159189
conv3-1	-66.297716	213373.701530
conv3-2	26.901056	176315.584343
conv3-3	271.583564	231860.095653
conv3-4	280.161073	912472.986495
conv4-1	-29.205451	1929054.816164
conv4-2	-436.253971	1889933.679637
conv4-3	-501.170109	1096032.013281
conv4-4	-559.733211	343271.077189
conv5-1	-199.225413	114754.968813
conv5-2	-119.662537	28587.189274
conv5-3	-63.662626	7147.346738
conv5-4	-45.421317	1570.898368
fc6	-8.993336	191.871532
fc7	-2.295338	13.615358
fc8	-0.001482	8.129878

network without changing the function computed by the network. From here on I refer to this method as *batch normalization retrofit* (BNR). BNR involves running a sample of inputs through the original network and using measured statistics of that sample at each layer to derive initial values for the $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$, population mean, and population variance parameters of the batch normalization layer such that the function computed by the new layer with batch normalization is identical to that computed by the original layer, the mean activation of the new layer is close to zero, and the variance of the activation of the new layer is close to one.

To motivate the derivation of BNR, let

$$\boldsymbol{x}_{\ell-1}^{T}\boldsymbol{W}_{\ell} + \boldsymbol{b}_{\ell} = \boldsymbol{x}_{\ell} \tag{24}$$

represent a simplified version of the computation performed by a layer of a network with L layers, where $\boldsymbol{x}_0$ is the input to the network and every $\boldsymbol{x}_{\ell}$ is the output of layer ℓ. The effects of nonlinearity and convolution are ignored, because they complicate the derivation of BNR without contributing insight. The input to layer ℓ is $\boldsymbol{x}_{\ell}$, $\boldsymbol{W}_{\ell}$ is the weight matrix, and $\boldsymbol{b}_{\ell}$ is the bias vector. I compute the average activation $\overline{\boldsymbol{x}}_{\ell}$ of the neurons in layer ℓ by taking a sum over some large number N of representative inputs:

$$\overline{\boldsymbol{x}}_\ell = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_\ell^{(i)} \tag{25}$$

and I compute a corresponding variance of the neurons in each layer ℓ. Square brackets in the subscript distinguish the vector index of D_ℓ -dimensional vector $\boldsymbol{s_\ell}$ from the layer index, ℓ.

$$s^2_{\ell[d]} = \frac{1}{N}\sum_{i=1}^{N} (\overline{x}_{\ell[d]} - x^{(i)}_{\ell[d]})^2 \tag{26}$$

The goal of BNR is to define functions $f_\ell(\cdot)$ to transform the inputs and outputs of each layer ℓ, and assign values to variables $\boldsymbol{\mu}_\ell$, $\boldsymbol{\sigma}_\ell$, $\boldsymbol{\gamma}_\ell$, and $\boldsymbol{\beta}_\ell$, which are the parameters needed to define the batch-normalized version of layer ℓ. The batch-normalized form of layer ℓ can be written as

$$\left(f_{\ell-1}(\boldsymbol{x}_{\ell-1})^T \boldsymbol{W}_\ell - \boldsymbol{\mu}_\ell\right) \odot \frac{1}{\boldsymbol{\sigma}_\ell + \boldsymbol{\epsilon}} \odot \boldsymbol{\gamma}_\ell + \boldsymbol{\beta}_\ell = f_\ell(\boldsymbol{x}_\ell) \tag{27}$$

in which the notational abbreviation $\frac{1}{\boldsymbol{\sigma}_\ell + \boldsymbol{\epsilon}}$ is used to indicate the numerically-stabilized element-wise reciprocal of the population standard deviation vector $\boldsymbol{\sigma}_\ell$. The definitions and variable assignments must satisfy the following requirements:

- The whole network in BN form must compute the same function as the original network.
- The values of the batch-normalization population mean, $\boldsymbol{\mu}_\ell$, and the batch-normalization population standard deviation, $\boldsymbol{\sigma}_\ell$, are small enough to mitigate the effects of gradient explosion or vanishing in training the multi-layer network.
- The values assigned to $\boldsymbol{\mu}_\ell$ and $\boldsymbol{\sigma}_\ell$ are the true population mean and standard deviation, respectively, of the transformed layer.

The requirement that the true population statistics are assigned to the population mean and population standard deviation variables used by batch normalization is what makes the problem of BNR interesting; ignoring this requirement would make it trivial to transform network layers to layers that superficially function like batch normalized layers in inference mode, but unless all of the variable assignments are consistent with the true statistics of the transformed layer, batch normalization would fail in training mode, where the population mean and variance variables are not applied, and mini-batch statistics recomputed at every training step are used in their places.

I define $f_\ell(\cdot)$, the transformation functions applied to the internal signals of each layer of the network, as follows:

$$f_\ell(\boldsymbol{x}_\ell) \equiv c_\ell \boldsymbol{x}_\ell \tag{28}$$

That is, the input to each layer in the network is transformed in an analogous way throughout the network, and the transformation (which is achieved by the previous layer) is linear. For each layer ℓ, c_ℓ is defined as:

$$c_\ell \equiv \frac{D_\ell}{\sum_{d=1}^{D_\ell} s_{\ell[d]}} \quad \text{if} \quad \ell > 0, \quad \text{otherwise} \quad 1 \tag{29}$$

The choice to transform signals in the network by a scalar multiplication rather than try to normalize the vector components independently was due to the observation that some pre-trained networks contain "dead" units that have a standard deviation of zero. In practice, the resulting approximation of batch statistics that results from this scalar transformation is close enough to the true statistics to keep things stable during training.

The batch-normalization parameters are then initialized as:

$$\boldsymbol{\mu}_\ell \equiv c_{\ell-1}(\overline{\boldsymbol{x}}_\ell - \boldsymbol{b}_\ell) \tag{30}$$

$$\boldsymbol{\sigma}_\ell \equiv c_{\ell-1}\boldsymbol{s}_\ell \tag{31}$$

$$\boldsymbol{\gamma}_\ell \equiv c_\ell \boldsymbol{s}_\ell \tag{32}$$

$$\boldsymbol{\beta}_\ell \equiv c_\ell \overline{\boldsymbol{x}}_\ell \tag{33}$$

Substituting these parameter definitions into Equation 27 and ignoring $\boldsymbol{\epsilon}$, a small value used for numerical stabilization, yields the original (non-batch-normalized) form of Equation 24, multiplied by c_ℓ. To satisfy the requirement that the network converted to batch-normalized form compute the same function as the original network, we can divide the output of the last stage by c_L. The requirement that the conversion mitigate the numerical problems of the non-batch-normalized network is empirically true for all networks tested; despite that the output of each layer is still not zero mean and unit variance, the constant scaling by c_ℓ at each layer results in optimization performance that is superior to that of the original network. It is straightforward to show that the mean and variance of the transformed network prior to scaling by $\boldsymbol{\gamma}_\ell$ and shifting by $\boldsymbol{\beta}_\ell$ are approximately equal to $\boldsymbol{\mu}_\ell$ and $\boldsymbol{\sigma}_\ell$ as defined above, satisfying the final requirement that ensures that the behavior of the batch-normalized layer will be correct in training mode.

The BNR method was immensely useful in making pre-trained models that did not include batch normalization more agile for transfer learning, by improving optimization performance, adding additional regularization, and permitting faster learning rates to be used. Table 4 exhibits the means and variances of internal signals after application of BNR.

Table 4: Means and variances of VGG19 internal signals after application of the BNR method

Signal Name	Mean	Variance
conv1-1	0.000112	0.069354
conv1-2	1.080532	9.761540
conv2-1	0.100548	3.258533
conv2-2	-0.207117	1.949663
conv3-1	-0.158944	1.225799
conv3-2	0.060037	0.881015
conv3-3	0.663601	1.384752
conv3-4	0.597524	4.151195
conv4-1	-0.031359	2.222103
conv4-2	-0.322884	1.035226
conv4-3	-0.371230	0.601322
conv4-4	-0.539286	0.318628
conv5-1	-0.341557	0.337199
conv5-2	-0.356291	0.253157
conv5-3	-0.379521	0.253580
conv5-4	-0.543064	0.222872
fc6	-0.230674	0.124072
fc7	-0.203100	0.071800
fc8	-0.000400	0.601449

A.5 Neural network prototyping design issues

TensorFlow (Abadi et al. [2015]) is a powerful tool for expressing computations and implementing machine learning systems. It has Python bindings for rapid prototyping, it supports heterogeneous and distributed computer architectures for scalability, and it is developed and maintained by machine-learning and distributed-systems experts. While TensorFlow was an invaluable tool in this work, it did not provide a complete solution to all problems I faced. I developed additional scaffolding and design patterns to facilitate my work, particularly to help isolate and control sources of error and variation when designing new models, and to manage large training datasets. In this section I describe several of the patterns and scaffolding components I found most useful for development, in the hope that they may be useful to others as well.

The challenge in engineering the components I needed in this project was to create strong enough abstractions to isolate the hyperparameters of the learning system from one another and from external factors such as hardware constraints and efficiency concerns. Section A.3 describes one especially problematic case in which I needed to modify a learning algorithm from the literature in order to accommodate the constraints of GPU hardware. There are many other ways in which hardware and learning models interact, and my development effort was guided by the following design criteria:

- Hardware configuration may affect speed of the learning algorithm but not other aspects, e.g., convergence or stability.

- Degree of data and model parallelism should be independently controllable without modification to the network specification.
- Specification of the training algorithm should not depend on hardware configuration. For example, the modules controlling whether we train a generative adversarial network, a classifier, or some other type of network should be well abstracted from the modules controlling what type of hardware is used and how it is configured.
- File-system read speed should not present a major bottleneck for training models.
- Saving and restoring models must be well separated from all other concerns, but,
  - saved models should be easy to manipulate and modify, and,
  - it must be straightforward to initialize a new model from saved weights originating from one or more previously trained models.

My scaffolding for training models is factored mainly into trainers, networks, and data pipeline elements. Network instances build segments of neural network within device contexts established by trainer instances. Trainer instances are responsible for setup and tear-down, and application of SGD optimization and other learning requirements of networks such as update of global statistics variables used in batch normalization. Data pipeline elements provide a common interface for retrieving a batch of data from buffers, disks, over a network, etc., so that the details of routing data into and out of neural network segments are easy to configure.

Trainers are described by a handful of classes that work like nouns, and several functions of type **class** $\longrightarrow$ **class** that work like adjectives. The noun-adjective idiom provides a convenient way to describe the purpose and hierarchical structure of a subclass compactly. For example, a trainer with a control algorithm for generative adversarial networks that uses data parallelism across 4 GPUs might be defined as:

```
MyTrainerSubclass = adversarial(multiGPU(4)(Trainer))
```

or, using Python's class-decorator syntax:

```
@adversarial
@multiGPU(4)
class MyTrainerSubclass2(Trainer):
    ...
```

The default Trainer expects to train a network with a single loss on one GPU. Other trainer classes exist for testing purposes. The list of adjectives includes:

- `adversarial`, which modifies the control algorithm to switch between training the generator and the adversary in a GAN, at intervals configurable via hyperparameters.
- `multiGPU`, which achieves data parallelism by creating many identical, parameter-sharing models on a multi-GPU system. The optimizing strategy is overridden to average the gradients from each GPU.

- `batchSequencing`, which divides the batch and averages gradients over several time steps to achieve a larger effective batch size for SGD; this class is necessary to avoid memory errors because TensorFlow cannot automatically split up batches that do not fit in GPU memory.
- `multiSegment`, which achieves model parallelism by splitting up neural networks across several devices to accommodate larger networks than would fit on a single device. It takes into account available DMA channels to configure the networks as efficiently as possible.

The multiSegment adjective works by dividing the number of separable units, N_u, of its network by the number of devices to use per network, N_d, and entering the network's *build* co-routine approximately N_u/N_d times within each TensorFlow device context created for each device. The *build* co-routine is responsible for ensuring that the network blocks wired up between each of the co-routine's yield statements are roughly consistent in size. The multiSegment adjective is used in conjunction with multiGPU to achieve model and data parallelism simultaneously.

The noun-adjective idiom for trainer modules makes it easy to design exhaustive tests for all meaningful combinations of trainer functionality, for example by training the same deterministically-initialized test network on many trainer configurations and ensuring that the results match. In deep learning experiments it is essential to have an extensively tested scaffolding like this, otherwise subtle inconsistencies introduced by the training methods will have compounding and unpredictable consequences.

To fulfill my design requirements for saved models, I relied mostly on TensorFlow's saving mechanism, with modifications to how saved models are restored. TensorFlow's saver, by default, throws an error if the parameters in the checkpoint file do not exactly match the variables in the current graph. I modified this behavior via a wrapper class to optimistically restore from checkpoint files; that is, restore variables when a mapping exists, discard unused variables in the checkpoint file, and initialize from scratch those variables that are not restored. This makes the exploratory process of designing new networks using pre-trained pieces of existing networks less cumbersome than in the default configuration. I also provided a convenient shorthand for migrating variables from one namespace to another, further reducing the burden of stored-model management for evolving models.

To address problems related to dataset management I designed a simple system of classes providing a pipeline abstraction that is easily configured to run in-process for fast transfer rate, or out of process for lower start-up latency from queue-filling when debugging a network. The data pipeline balances CPU-intensive post-processing such as image-patch rotation and scaling with file-system waits and GPU intensive work to avoid idle time. This balancing alone was not enough to solve the latency problems: profiling during training of my depth-estimation network with simple FIFO queues revealed that disk read time accounted for 93% of program execution time. I resolved this by reading training data into a large ring buffer in memory. The dataset is constantly sampled randomly and added to the ring buffer while the oldest examples are removed, and training batches are produced by randomly sampling from the memory-resident buffer. To mitigate overfitting this slowly-varying window, the pipeline applies data augmentation such as random cropping, scaling, and rotation whenever examples are retrieved from the buffer, as described in Section 5.6.1. In expanded form, the NYU Depth dataset (Silberman et al. [2012]) used for training is over 500GB in size and a memory buffer size of 22GB produced initial loss curves similar to those obtained from training on the whole dataset.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Juan D. Adarve and Robert Mahony. A Filter Formulation for Computing Real Time Optical Flow. *IEEE Robotics and Automation Letters*, 1(2):1192–1199, July 2016. ISSN 2377-3766. doi: 10.1109/LRA.2016.2532928.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015. URL http://www.cv-foundation.org/openaccess/content_iccv_2015/html/Antol_VQA_Visual_Question_ICCV_2015_paper.html.

Andrei Barbu, Aaron Michaux, Siddharth Narayanaswamy, and Jeffrey Mark Siskind. Simultaneous Object Detection, Tracking, and Event Recognition. *arXiv:1204.2741 [cs]*, April 2012. URL http://arxiv.org/abs/1204.2741. arXiv: 1204.2741.

Daniel Paul Barrett, Andrei Barbu, N. Siddharth, and Jeffrey Mark Siskind. Saying What You're Looking For: Linguistics Meets Video Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2069–2081, October 2016. ISSN 0162-8828, 2160-9292. doi: 10.1109/TPAMI.2015.2505297. URL http://ieeexplore.ieee.org/document/7346469/.

Jacob Stuart Michael Beal. *Learning by learning to communicate*. PhD thesis, Massachusetts Institute of Technology, 2007.

Guy Ben-Yosef, Liav Assif, Daniel Harari, and Shimon Ullman. A model for full local image interpretation. In *CogSci*, 2015. URL https://pdfs.semanticscholar.org/ca8c/3c616b757b77eba6d8689d5f1ca57d6b81ca.pdf.

Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, and Bert Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0130266922.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. *arXiv:1412.7062 [cs]*, December 2014. URL http://arxiv.org/abs/1412.7062. arXiv: 1412.7062.

Michael Coen. *Multimodal Dynamics: Self-Supervised Learning in Perceptual and Motor Systems*. PhD thesis, Massachusetts Institute of Technology, May 2006.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

Steve Dent. Google's AI is getting really good at captioning photos. *engadget.com*, Sep 2016. URL https://www.engadget.com/2016/09/23/googles-ai-is-getting-really-good-at-captioning-photos/.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial Feature Learning. *arXiv:1605.09782 [cs, stat]*, May 2016. URL http://arxiv.org/abs/1605.09782. arXiv: 1605.09782.

David Eigen and Rob Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. *arXiv:1411.4734 [cs]*, November 2014. URL http://arxiv.org/abs/1411.4734. arXiv: 1411.4734.

David Eigen, Christian Puhrsch, and Rob Fergus. Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. *arXiv:1406.2283 [cs]*, June 2014. URL http://arxiv.org/abs/1406.2283. arXiv: 1406.2283.

Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, MA; London, May 1998. ISBN 978-0-262-06197-1. URL http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=8106.

Yoav Freund and Robert E.gir Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X.

Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, December 2015. doi: 10.1109/ICCV.2015.169.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014a. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, December 2014b. URL http://arxiv.org/abs/1412.6572. arXiv: 1412.6572.

Noah Goodman, Vikash Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. *arXiv:1206.3255 [cs]*, June 2012. URL http://arxiv.org/abs/1206.3255. arXiv: 1206.3255.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.

Geoffrey Hinton. A practical guide to training restricted Boltzmann machines. Technical report, Department of Computer Science, University of Toronto, Toronto, Canada, 2010.

Geoffrey E. Hinton. *Relaxation and its role in vision*. PhD thesis, Edinburgh University, 1978.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):502–507, July 2006. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1129198. URL http://www.sciencemag.org/cgi/doi/10.1126/science.1129198.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.2006.18.7.1527.

Derek Hoiem, Alexei A. Efros, and Martial Hebert. Automatic photo pop-up. *ACM transactions on graphics (TOG)*, 24(3):577–584, 2005.

Berthold K. P. Horn and Michael J. Brooks. *Shape from Shading*. MIT Press, Cambridge, MA, 1989.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed 7/4/2017].

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Pang Wei Koh and Percy Liang. Understanding Black-box Predictions via Influence Functions. *arXiv:1703.04730 [cs, stat]*, March 2017. URL http://arxiv.org/abs/1703.04730. arXiv: 1703.04730.

Peter Kovesi. Good Colour Maps: How to Design Them. *arXiv:1509.03700 [cs]*, September 2015. URL http://arxiv.org/abs/1509.03700. arXiv: 1509.03700.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Yevhen Kuznietsov, Jörg Stückler, and Bastian Leibe. Semi-Supervised Deep Learning for Monocular Depth Map Prediction. *arXiv:1702.02706 [cs]*, February 2017. URL http://arxiv.org/abs/1702.02706. arXiv: 1702.02706.

L'ubor Ladický, Jianbo Shi, and Marc Pollefeys. Pulling Things out of Perspective. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 89–96, June 2014. doi: 10.1109/CVPR.2014.19.

Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper Depth Prediction with Fully Convolutional Residual Networks. *arXiv:1606.00373 [cs]*, June 2016. URL http://arxiv.org/abs/1606.00373. arXiv: 1606.00373.

Bo Li, Chunhua Shen, Yuchao Dai, A. van den Hengel, and Mingyi He. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1119–1127, June 2015. doi: 10.1109/CVPR.2015.7298715.

Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2024–2039, October 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2505283.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. URL http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html.

Jack M. Loomis. Looking down is looking up. *Nature*, 414(6860):155–156, 2001.

John Markoff. Researchers Announce Advance in Image-Recognition Software. *The New York Times*, Nov 2014. URL https://www.nytimes.com/2014/11/18/science/researchers-announce-breakthrough-in-content-recognition-software.html.

David Marr. *Vision: A Computational Investigation Into the Human Representation and Processing of Visual Information*. MIT Press, 2010. ISBN 9780262514620.

David A. McAllester. A three valued truth maintenance system. Technical report, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1978.

Harry McGurk and John MacDonald. Hearing lips and seeing voices. *Nature*, 264:746, December 1976. URL http://dx.doi.org/10.1038/264746a0.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv:1610.08401 [cs, stat]*, October 2016. URL http://arxiv.org/abs/1610.08401. arXiv: 1610.08401.

Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. UFLDL Tutorial, 2013. URL http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 689–696, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Ngiam_399.pdf.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 427–436. IEEE, 2015.

Teng Leng Ooi, Bing Wu, and Zijiang J. He. Distance determined by the angular declination below the horizon. *Nature*, 414(6860):197–200, 2001.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. *arXiv:1211.5063 [cs]*, November 2012. URL http://arxiv.org/abs/1211.5063. arXiv: 1211.5063.

Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning Features by Watching Objects Move. *arXiv:1612.06370 [cs, stat]*, December 2016. URL http://arxiv.org/abs/1612.06370. arXiv: 1612.06370.

Alexey Radul. *Propagation Networks: A Flexible and Expressive Substrate for Computation*. PhD thesis, Massachusetts Institute of Technology, 2009.

Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1164–1172, 2015.

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, 2015a. doi: 10.1007/s11263-015-0816-y.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015b.

Ashutosh Saxena, Min Sun, and Andrew Y. Ng. Make3D: Depth Perception from a Single Still Image. In *AAAI*, pages 1571–1576, 2008.

Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv:1312.6229 [cs]*, December 2013. URL http://arxiv.org/abs/1312.6229. arXiv: 1312.6229.

Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *arXiv:1605.06211 [cs]*, May 2016. URL http://arxiv.org/abs/1605.06211. arXiv: 1605.06211.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. *Computer Vision–ECCV 2012*, pages 746–760, 2012.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, September 2014. URL http://arxiv.org/abs/1409.1556. arXiv: 1409.1556.

Jeffrey Siskind and Quaid Morris. A maximum-likelihood approach to visual event classification. *Computer Vision—ECCV'96*, pages 347–360, 1996. URL http://www.springerlink.com/index/7008747l42073u0q.pdf.

Jeffrey Mark Siskind. Grounding language in perception. In *Integration of Natural Language and Vision Processing*, pages 207–227. Springer, 1995. URL http://link.springer.com/chapter/10.1007/978-94-011-0273-5_12.

Nitish Srivastava and Ruslan R. Salakhutdinov. Multimodal learning with deep Boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012. URL http://papers.nips.cc/paper/4683-multimodal-learning-with-deep-boltzmann-machines.

Richard M. Stallman and Gerald J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135 – 196, 1977. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(77)90029-7. URL http://www.sciencedirect.com/science/article/pii/0004370277900297.

Chris Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, page 252 Vol. 2, 1999. doi: 10.1109/CVPR.1999.784637.

Gerald Jay Sussman and Alexey Radul. The Art of the Propagator. Technical Report MIT-CSAIL-TR-2009-002, MIT Computer Science and Artificial Intelligence Laboratory, 2009.

Gerald Jay Sussman and Guy Lewis Steele. CONSTRAINTS—A language for expressing almost-hierarchical descriptions. *Artificial Intelligence*, 14(1):1–39, 1980.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Shimon Ullman. Sequence-seeking and counter streams: A model for information processing in the cortex. Memo 1311, Artificial Intelligence Laboratory, MIT, 1991.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518. IEEE, 2001.

Andrew J. Viterbi. Convolutional Codes and Their Performance in Communication Systems. *IEEE Transactions on Communication Technology*, 19(5):751–772, October 1971. ISSN 0018-9332. doi: 10.1109/TCOM.1971.1090700.

David L. Waltz. *Generating Semantic Descriptions From Drawings of Scenes With Shadows*. PhD thesis, Cambridge MA, 1972.

Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. DeepFlow: Large Displacement Optical Flow with Deep Matching. pages 1385–1392. IEEE, December 2013. ISBN 978-1-4799-2840-8. doi: 10.1109/ICCV.2013.175. URL http://ieeexplore.ieee.org/document/6751282/.

Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, third edition, 1992. ISBN 0201533774.

Patrick Henry Winston. The Genesis story understanding and story telling system: A 21^{st} century step toward artificial intelligence. Memo 019, Center for Brains Minds and Machines, MIT, 2014.

Patrick Henry Winston and Dylan Holmes. The Genesis manifesto: Story understanding and human intelligence. In preparation., 2018.

Ramin Zabih. Dependency-directed backtracking in non-deterministic Scheme. 1988.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv:1611.03530 [cs]*, November 2016a. URL http://arxiv.org/abs/1611.03530. arXiv: 1611.03530.

Richard Zhang, Phillip Isola, and Alexei A. Efros. Split-Brain Autoencoders: Unsupervised Learning by Cross-Channel Prediction. *arXiv:1611.09842 [cs]*, November 2016b. URL http://arxiv.org/abs/1611.09842. arXiv: 1611.09842.

Zoran Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 28–31 Vol.2, August 2004. doi: 10.1109/ICPR.2004.1333992.

Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773–780, May 2006. ISSN 01678655. doi: 10.1016/j.patrec.2005.11.005. URL http://linkinghub.elsevier.com/retrieve/pii/S0167865505003521.

Daniel Zoran, Phillip Isola, Dilip Krishnan, and William T. Freeman. Learning ordinal relationships for mid-level vision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 388–396, 2015. URL http://www.cv-foundation.org/openaccess/content_iccv_2015/html/Zoran_Learning_Ordinal_Relationships_ICCV_2015_paper.html.