Sculpting Representations for Deep Learning

by

Oren Rippel

B.Sc. Hons, University of British Columbia (2010)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .... Signature redacted ....
Department of Mathematics
April 22, 2016

Certified by Signature redacted ....
Ryan P. Adams
Professor
Thesis Supervisor

Certified by. Signature redacted ....
Ankur Moitra
Professor
Thesis Supervisor

Accepted by Signature redacted ....
Jonathan A. Kelner
Chair, Department Committee on Graduate Studies

Sculpting Representations for Deep Learning

by

Oren Rippel

Submitted to the Department of Mathematics
on April 22, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In machine learning, the choice of space in which to represent our data is of vital importance to their effective and efficient analysis. In this thesis, we develop approaches to address a number of problems in representation learning. We employ deep learning as means of sculpting our representations, and also develop improved representations for deep learning models.

We present contributions that are based on five papers and make progress in several different research directions. First, we present techniques which leverage spatial and relational structure to achieve greater computational efficiency of model optimization and query retrieval. This allows us to train distance metric learning models 5-30 times faster; optimize convolutional neural networks 2-5 times faster; perform content-based image retrieval hundreds of times faster on codes hundreds of times longer than feasible before; and improve the complexity of Bayesian optimization to linear in the number of observations in contrast to the cubic dependence in its naïve Gaussian process formulation.

Furthermore, we introduce ideas to facilitate preservation of relevant information within the learned representations, and demonstrate this leads to improved supervision results. Our approaches achieve state-of-the-art classification and transfer learning performance on a number of well-known machine learning benchmarks.

In addition, while deep learning models are able to discover structure in high dimensional input domains, they only offer implicit probabilistic descriptions. We develop an algorithm to enable probabilistic interpretability of deep representations. It constructs a transformation to a representation space under which the map of the distribution is approximately factorized and has known marginals. This allows tractable density estimation and inference within this alternate domain.

Thesis Supervisor: Ryan P. Adams
Title: Professor

Thesis Supervisor: Ankur Moitra
Title: Professor

Acknowledgments

I would like to express my deep appreciation to a number of people who have had strong impact on my life in the last few years.

First, I am indebted to my advisor, Ryan Adams. You shaped the way I think — not only with respect to machine learning, but also beyond academia. Your guidance enabled me to reach my full potential, and my best interests have always been your primary concern. Throughout my Ph.D., I have many times over concluded that I couldn't have wished for a better advisor.

I would like to thank my sister Noa, and my parents Dafna and Doron. You have dedicated so much of your lives to enable me to actualize mine. Without your love, support, and burekasim, I wouldn't be who I am today.

I am grateful to Manohar Paluri, Lubomir Bourdev and Piotr Dollar at Facebook AI Research and Applied Machine Learning. I learned a great deal working at Facebook, and my time with you was very meaningful and formative to me.

I would also like to express my gratitude to Ankur Moitra and Jonathan Kelner for serving on my thesis committee. I would especially like to thank Ankur for valuable advice and helpful discussions.

I would like to thank my collaborators and friends Jasper Snoek, Michael Gelbart, David Duvenaud, Matt Johnson, Diana Cai, Robert Nishihara, Adrian Jinich, Scott Linderman, Polina Golland, Yakir Reshef, Jon Malmaud, Kevin Swersky, Hugo Larochelle, Anna Huang, James Zou, Finale Doshi-Velez, Dougal Maclaurin, and Prabhat. I learned a great deal working with you, and thoroughly enjoyed the time spent with you.

I am also thankful to Gershon Ben Keren and the entirety of the Krav Maga Yashir school for unparalleled companionship. After long days of work, training with you has never failed to brighten me up.

Lastly but most importantly, I would like to thank Tania Saade. In the highs and in the lows, you never stopped believing in me. Without your unwavering support, encouragement, and warmth, this thesis would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The raw representations of my input data are perfectly suitable for direct analysis."

— No one, ever

Machine learning is the science of pattern recognition in data. It explores algorithms that automatically *learn* from data, as opposed to being explicitly programmed to follow hard-coded rules. Typical problems in the field include discovering underlying structure in an input set of examples, and leveraging it to make predictions or decisions.

Our success in performing any such task critically depends on the choice of space in which to represent our inputs. However, the original input formats of our data are often far from suitable for straightforward and efficient extraction of pertinent information. Consider, for example, the space of 1024×1024 images in their pixel-RGB formats. To start, this large dimensionality of $\approx 3,100,000$ coordinates prohibits efficient processing. Moreover, this domain features significant redundancy due to variation in directions not relevant to our task at hand: namely, natural images make an infinitesimal fraction of all possible combinations of pixel values. This results in conflation of signal and noise, which obfuscates analysis.

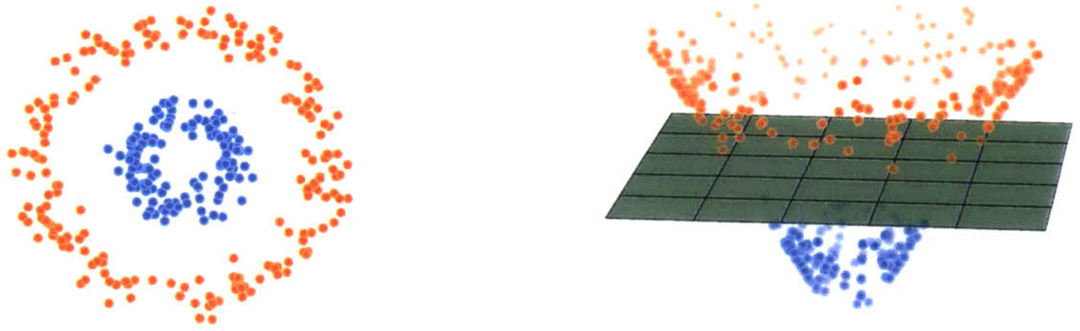

(a) Original input representation $\mathbf{x}$. (b) Alternative representation $\mathbf{z}$.

Figure 1-1: Toy example of exploitation of input structure for linear decision boundaries. Presented are distributions of samples generated from the toy model in Equation (1.1) under different input representations. The colors correspond to the two different classes. **(a)** Scatter plot of $\mathbf{x}$ in raw input form. The two classes cannot be separated by a linear classifier. **(b)** Scatter plot of $\mathbf{z} = [\mathbf{x}, \|\mathbf{x}\|_2]^T$. The classes can be separated with a linear classifier — namely, a horizontal plane.

On the other side of the spectrum, in our *ideal world* unplagued with the aforementioned challenges, our hope is to conquer our problem at hand with very simple approaches. For instance, if we were able to solve a categorization problem with a linear classifier, we would be able to reap its benefits of efficiency, scalability, and availability of theoretical guarantees. However, our raw data are often not amenable to such simple solution.

As such, we are motivated to transform our inputs into alternative *representations* embedded in a domain which empowers our tools of analysis. In real life, we will clearly not always have a-priori insight to enable acute simplification; however, by designing a suitable map to an alternative representation space, we can compromise on a middle ground that will greatly facilitate our task to follow.

To solidify this discussion, consider, for example, the toy classification problem of predicting binary class $c \in \{0, 1\}$ from label $\mathbf{x} \in \mathbb{R}^2$, where examples are generated from the ground truth model

$$\begin{aligned}
c &\sim \text{Bernoulli}\,(1/2) \\
\theta &\sim \text{Uniform}\,([0, 2\pi)) \\
\varepsilon &\sim \mathcal{N}(0, \sigma^2)\,, \qquad \sigma^2 \ll R_0, R_1 \\
r &= R_0 \mathbb{1}_{c=0} + R_1 \mathbb{1}_{c=1} + \varepsilon \\
\mathbf{x} &= [r \sin\theta,\; r\cos\theta]^T\;. \qquad (1.1)
\end{aligned}$$

Here, $\mathbb{1}_A$ is an indicator function for event A. Equation (1.1) has a simple intuitive interpretation: for each example, we flip a coin c to generate its class; $\mathbf{x}$ is then sampled as a vector with a uniform angle, and a particular radius R_0 or R_1 as function of the class. The radius is then perturbed with white noise ε.

In Figure 1-1(a), we observe 100 input-output pairs sampled from this model. From this plot, it can be seen that the two classes cannot be simply separated with a linear decision boundary. However, let us assume we had the prescience to augment our input representation with the transformation $r(\mathbf{x}) = \|\mathbf{x}\|_2$ as its third dimension. Under these alternative representations, we would be able to easily separate the classes with a horizontal plane. This can be observed in Figure 1-1(b).

In some cases, such as in this example, useful representations can be fashioned via projections onto hand-crafted features. However, the success of such an approach hinges on our ability to discover structure in our inputs, and manually design a suitable basis tailored to leverage it. Our toy example is vastly oversimplified: in more realistic scenarios, our inputs will be of high dimensionality, and their distribution too complex to identify dependence by simple visualization.

Alternatively, the transformation to representation space can be learned adaptively as function of the input data; for instance, a simple and well-known adaptive approach is Principal Component Analysis, which performs dimensionality reduction via a linear projection optimized to preserve variance. However, given more complex representational requirements, it is difficult to manually devise an appropriate map to a space which conforms to them. Moreover, by fashioning alternative representations for our inputs, our intent is to facilitate their subsequent use for prediction. As such, we are interested to inform the algorithm used to construct them of their deployment to follow in order optimize them accordingly.

A powerful approach, then, is to *automatically learn our representations*: we formalize our representational and predictive demands into a single objective, which can then be optimized over a flexible class of parametrized transformations. This enables us to avoid explicit feature engineering as a separate step; indeed, approaches to feature learning have often found representations that outperform their hand-crafted counterparts (e.g., LeCun & Bengio, 1995; Hinton & Salakhutdinov, 2006; Vincent et al., 2010; Coates et al., 2011).

One natural and powerful framework for the construction of such parametrized classes is *deep learning*. Deep learning is a subfield of machine learning, characterized by its employment of hierarchical feature extraction (LeCun et al., 2015; Bengio, 2009; LeCun et al., 1989). Deep learning models constitute rich, expressive and interpretable parametrizations. Specifically, their modularity and compositionality render them particularly effective for representation learning. That is, they enable us to jointly train the end-to-end pipeline composed of the extraction of features and their subsequent use for prediction.

In this thesis, we present contributions to representation learning that are based on five papers; a complete summary of these can be found in Section 1.1 below. We employ deep learning as means of sculpting our representations, and also develop more salient representations *for* deep learning models. We develop approaches that make progress in several different research directions. Briefly, these include:

Reduction of computational complexity. We develop algorithms to mold the representation space for convenient descriptions by specialized data structures which enable manipulations of lower computational complexity. For example, by constructing a representation space suitable for description by a binary tree, we perform content-based image retrieval hundreds of times faster on codes hundreds of times longer than feasible before. In a different work, we improve the complexity of Bayesian optimization to linear in the number of observations contrast to the cubic dependence in its naïve Gaussian process formulation.

INCREASED EFFICIENCY OF MODEL OPTIMIZATION. We design alternative representations to leverage spatial structure in convolutional neural networks, which allows us to optimize them 2-5 times faster. In another work, we exploit representational similarity structure, which allows us to train distance metric learning models 5-30 times faster.

IMPROVED SUPERVISION AND TRANSFER LEARNING. We introduce ideas to facilitate preservation of relevant information within the learned representations. We demonstrate in a number of instances that this leads to improved supervision results. Our approaches achieve state-of-the-art classification and transfer learning performance on an array of well-known machine learning benchmarks.

PROBABILISTIC INTERPRETABILITY. The distribution of data in their raw input space tends to be multi-modal and ill-conditioned, and of dimensionality prohibitively large for effective probabilistic analysis. On the other end of the spectrum, deep learning models are able to discover structure in such domains, but only offer implicit probabilistic descriptions. We propose an algorithm to construct a transformation to a representation space under which the map of the distribution is approximately factorized and has known marginals. This allows tractable density estimation and inference within this alternate domain.

1.1 Contributions and Outline of Thesis

1.1.1 Chapter 3: Learning Ordered Representations with Nested Dropout

In this work, we present results on ordered representations of data in which different dimensions have different degrees of importance. To learn these representations we introduce *nested dropout*, a procedure for stochastically removing coherent nested sets of hidden units in a neural network. We first present a sequence of theoretical results for the special case of a semi-linear autoencoder. We rigorously show that the application of nested dropout enforces identifiability of the units, which leads to an exact equivalence with PCA.

We then extend the algorithm to deep models and demonstrate the relevance of ordered representations to a number of applications. Specifically, we use the ordered property of the learned codes to construct hash-based data structures that permit very fast retrieval, achieving retrieval in time logarithmic in the database size and independent of the dimensionality of the representation. This allows codes that are hundreds of times longer than currently feasible for retrieval. We therefore avoid the diminished quality associated with short codes, while still performing retrieval that is competitive in speed with existing methods.

We also show that ordered representations are a promising way to learn adaptive compression for efficient online data reconstruction.

This chapter is based on the following paper:

Rippel, Oren, Gelbart, Michael A., and Adams, Ryan P. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, 2014

1.1.2 Chapter 4: Metric Learning with Adaptive Density Discrimination

Distance metric learning (DML) approaches learn a transformation to a representation space where distance is in correspondence with a predefined notion of similarity. While such models offer a number of compelling benefits, it has been difficult for these to compete with modern classification algorithms in performance and even in feature extraction.

In this work, we propose a novel approach explicitly designed to address a number of subtle yet important issues which have stymied earlier DML algorithms. It maintains an explicit model of the distributions of the different classes in representation space. It then employs this knowledge to adaptively assess similarity, and achieve local discrimination by penalizing class distribution overlap.

We demonstrate the effectiveness of this idea on several tasks. Our approach achieves state-of-the-art classification results on a number of fine-grained visual recognition datasets, surpassing the standard softmax classifier and outperforming triplet loss by a relative margin of 30-40%. In terms of computational performance, it alleviates training inefficiencies in the traditional triplet loss, reaching the same error in 5-30 times fewer iterations. Beyond classification, we further validate the saliency of the learnt representations via their attribute concentration and hierarchy recovery properties, achieving 10-25% relative gains on the softmax classifier and 25-50% on triplet loss in these tasks.

This chapter is based on the following paper:

Rippel, Oren, Paluri, Manohar, Dollar, Piotr, and Bourdev, Lubomir. Metric learning with adaptive density discrimination. In *International Conference on Learning Representations*, 2016

1.1.3 Chapter 5: Spectral Representations for Convolutional Neural Networks

Discrete Fourier transforms provide a significant speedup in the computation of convolutions in deep learning. In this work, we demonstrate that, beyond its advantages for efficient computation, the spectral domain also provides a powerful representation in which to model and train convolutional neural networks (CNNs).

We employ spectral representations to introduce a number of innovations to CNN design. First, we propose *spectral pooling*, which performs dimensionality reduction by truncating the representation in the frequency domain. This approach preserves considerably more information per parameter than other pooling strategies and enables flexibility in the choice of pooling output dimensionality. This representation also enables a new form of stochastic regularization by randomized modification of resolution. We show that these methods achieve competitive results on classification and approximation tasks, without using any dropout or max-pooling.

Finally, we demonstrate the effectiveness of complex-coefficient spectral parameterization of convolutional filters. While this leaves the underlying model unchanged, it results in a representation that greatly facilitates optimization. We observe on a variety of popular CNN configurations that this leads to significantly faster convergence during training.

This chapter is based on the following paper:

Rippel, Oren, Snoek, Jasper, and Adams, Ryan P. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, 2015

1.1.4 Chapter 6: Scalable Bayesian Optimization Using Deep Neural Networks

Bayesian optimization is an effective methodology for the global optimization of functions with expensive evaluations. It relies on querying a distribution over functions defined by a relatively cheap surrogate model. An accurate model for this distribution over functions is critical to the effectiveness of the approach, and is typically fit using Gaussian processes (GPs). However, since GPs scale cubically with the number of observations, it has been challenging to handle objectives whose optimization requires many evaluations, and as such, massively parallelizing the optimization.

In this work, we explore the use of neural networks as an alternative to GPs to model distributions over functions. We show that performing adaptive basis function regression with a neural network as the parametric form performs competitively with state-of-the-art GP-based approaches, but scales linearly with the number of data rather than cubically. This allows us to achieve a previously intractable degree of parallelism, which we apply to large scale hyperparameter optimization, rapidly finding competitive models on benchmark object recognition tasks using convolutional networks, and image caption generation using neural language models.

This chapter is based on the following paper:

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Md. Mostofa Ali, Prabhat, and Adams, Ryan P. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015

1.1.5 Chapter 7: High-Dimensional Probability Estimation with Deep Density Models

One of the fundamental problems in machine learning is the estimation of a probability distribution from data. Many techniques have been proposed to study the structure of data, most often building around the assumption that observations lie on a lower-dimensional manifold of high probability. It has been more difficult, however, to exploit this insight to build explicit, tractable density models for high-dimensional data.

In this work, we introduce the *deep density model* (DDM), a new approach to density estimation. We exploit insights from deep learning to construct a bijective map to a representation space, under which the transformation of the distribution of the data is approximately factorized and has identical and known marginal densities.

The simplicity of the latent distribution under the model allows us to feasibly explore it, and the invertibility of the map to characterize contraction of measure across it. This enables us to compute normalized densities for out-of-sample data. This combination of tractability and flexibility allows us to tackle a variety of probabilistic tasks on high-dimensional datasets, including: rapid computation of normalized densities at test-time without evaluating a partition function; generation of samples without MCMC; and characterization of the joint entropy of the data.

This chapter is based on the following paper:

Rippel, Oren and Adams, Ryan P. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013

Chapter 2

Background and Notation

In this chapter, we provide a general introduction to deep learning. We start by describing the philosophy of model construction, and proceed to describe particular architectural choices. The concepts are not theoretically challenging, but there is field-specific jargon that requires translation.

Deep learning is a subfield of machine learning, defined by its sequential projections onto learned bases (LeCun et al., 2015; Bengio, 2009; LeCun et al., 1989). In the last few years, deep learning has undergone a renaissance: even though neural networks have existed for several decades, only recently has computational power reached a caliber sufficient to implement them at scale. Since the field's resuscitation, deep learning models have achieved competitive empirical performance on an array of notorious benchmarks across a spectrum of domains in science and industry. These include tasks in visual recognition, object detection, video, audio and text analysis, and more (e.g., Glorot et al., 2011a; Hinton et al., 2012a; Krizhevsky et al., 2012; Ouyang & Wang, 2013; Taigman et al., 2014; Szegedy et al., 2015). In this chapter, we will demystify on some of the factors behind the success of this framework.

2.1 General Model Formulation

The particular model architecture employed is specialized to suit the problem it seeks to address. For specificity and simplicity of exposition, we consider as an example a generic recognition task; the ideas presented generalize to other forms of input and other objectives beyond simple classification.

INPUT DATA Let us assume a training set consisting of N input-label pairs $\mathcal{D} := \{\mathbf{x}_n, y_n\}_{n=1}^N \subset \mathbb{R}^{K_0} \times \{1, \ldots, C\}$, that is, with datum dimensionality K_0 and ground truth label belonging to one of C classes. In real-life settings, N may vary from thousands to tens of millions of examples, and C from 2 classes in the case of binary classification to hundreds of thousands.

MODEL CONSTRUCTION We proceed to propagate an input $\mathbf{x}$ through a sequence of *layers*, corresponding to transformations $\mathbf{f}^{(m)}(\cdot;\, \boldsymbol{\Theta}_m) : \mathbb{R}^{K_{m-1}} \to \mathbb{R}^{K_m},\ m = 1, \ldots, M$:

$$\mathbf{r}^{(m)} := \mathbf{f}^{(m)}(\mathbf{r}^{(m-1)};\, \boldsymbol{\Theta}_m) \qquad \forall m = 1, \ldots, M \tag{2.1}$$

where we denote $\mathbf{r}^{(0)} := \mathbf{x}$. For each layer $m = 1, \ldots, M$, particular output dimensions $f_k^{(m)}$ for $k = 1, \ldots, K_m$ are referred to as *units*, and instantiations $r_k^{(m)}$ of these for particular examples as *activations*. The parameters for layer m, denoted $\boldsymbol{\Theta}_m \in \mathbb{R}^{D_m}$, are learned during training of the model. In total, a model can have as many as billions of parameters. The transformations $\mathbf{f}_m(\cdot;\cdot)$ take a variety of functional forms, such as linear transformations, where the parameters correspond to a learned matrix, or sigmoidal maps which carry no learnable parameters at all. We discuss in detail various common choices for these in Section 2.2.

OBJECTIVE DEFINITION Ultimately, following propagation through the network, in the case of classification, the output $\mathbf{r}^{(M)}$ is then contracted to a scalar via some *loss function* $\ell(\cdot, y) : \mathbb{R}^{K_M} \to \mathbb{R}$ which characterizes the quality of the prediction performed by the model with respect to the ground truth label y.

The objective function to be optimized, then, is taken (in this instance) as the average of all individual penalties

$$\mathscr{L}(\Theta_1, \ldots, \Theta_M) = \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{r}_n^{(M)}, y_n) \tag{2.2}$$

and its optimal solution is given by

$$\Theta_1^\star, \ldots, \Theta_M^\star = \arg \min_{\Theta_1, \ldots, \Theta_M} \mathscr{L}(\Theta_1, \ldots, \Theta_M) \ . \tag{2.3}$$

Much like the architecture itself, there are many choices for the evaluation metric. We discuss these in Section 2.2.

2.1.1 Intuitive interpretation

Each layer in the network can be considered a simple building block, which either independently transforms each activation, or mixes between them. As such, the end-to-end network can be loosely perceived to apply a sequence of logic operators. The level of abstraction, then, increases as function of layer index.

In a convolutional neural network applied to image data, for example, this interpretation can be observed more concretely. Let us, again, consider a generic visual recognition task. Assume that the input is processed by repeated alternation between applications of convolutional layers and nonlinearities. In this case, the first layer of the model tends to learn *edge detectors*, often colloquially referred to as "Gabor filters". While filtering of an image by these does not provide any prediction of its contents, it does provide representations more suitable for further processing. The second layer, in turn, is then able to aggregate spatial proximity of edges to detect *parts* of objects. The third layer, then, aggregates parts of objects to infer identity. For example, in a binary classification problem where we seek to determine whether each input is a cow or a tractor, this layer can be empirically observed to reason that "if I detect 4 legs, a tail, and a head, this is a cow with high probability".

Note, on the other hand, that if we instead sought to distinguish between cows and cheetahs, the above condition would not suffice to uniquely characterize cows.

To achieve discrimination, then, this last layer, then, would adapt to further evaluate whether the input is also yellow and spotted. This observation accentuates a significant appeal in our framework: our pipeline of filtering and decision-making is learned *completely autonomously* as function of the task at hand. It is true that the hierarchical filtering procedure introduced indeed constitutes an inductive bias hard-coded into the model setup; the particular transformations, however, are optimized to achieve the specific goal defined by $\ell(\cdot, \cdot)$. As such, deep learning can be perceived as *adaptive basis function regression.*

2.2 Typical Architecture Choices

We now briefly introduce different types of layers that will be employed throughout this thesis.

There are many choices for each component of the architecture, it is often not clear what the optimal arrangement is. There is a substantial body of work exploring these choices; however, the optimal solution varies substantially across domain applications and problems, and the search space is often prohibitively large — even for merely tens of hyperparameters explored. This motivates the need for principled hyperparameter optimization techniques, such as Bayesian Optimization. See Chapter 6 for more information.

Neural network architecture components tend to fall into five principal categories: parametrized layers, pooling layers, nonlinearities, stochastic regularization layers and losses.

Parametrized layers

These layers include a set of parameters that are to be learned during optimization.

FULLY-CONNECTED This linear layer simply computes a matrix product: $\mathbf{f}(\mathbf{x}; \mathbf{W}) = \mathbf{W}\mathbf{x}$ for learnable parameters $\mathbf{W} \in \mathbb{R}^{K \times D}$. The K rows of this matrix are known as *filters.* This layer is a main building block of deep models due to its wide applicability.

However, it should be employed along with strong regularization, as it tends to result in overfitting.

CONVOLUTIONAL Networks that include this layer are known as *convolutional neural networks* (CNNs). This is a special case of a fully-connected layer, which allows exploiting spatial structure in images, temporal structure in audio or text, spatiotemporal structure in videos, and so on. In its 2-dimensional map size specialization, for example, this layer has K learnable filters, each a tensor $\mathbf{W}_k \in \mathbb{R}^{C \times P \times Q}$ for C input channels, and map size $P \times Q$. Each such filter is then convolved across every input: it is slid across the input map with some fixed stride S (for simplicity of exposition assumed to be the same across both dimensions), and at each point contracts across its spatial window as well as all channels:

$$\begin{aligned} \mathbf{f}(\mathbf{x}, \mathbf{W}_k)_{hw} &= [\mathbf{x} * \mathbf{W}_k]_{hw} & (2.4) \\ &= \sum_{c=1}^{C} \sum_{p=1}^{P} \sum_{q=1}^{Q} W_{cpq} \, x_{c,\, hS+p,\, wS+q} & (2.5) \end{aligned}$$

for an input $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$, and $h = 1, \ldots, \lfloor \frac{H-P}{S} \rfloor + 1, w = 1, \ldots, \lfloor \frac{W-Q}{S} \rfloor + 1$. The above is given for zero padding around the input, but can be easily generalized to include it.

Note that these filters have clear spatial coherence: as such, it is appealing to learn these in their *spectral representations.* In this thesis, we introduce spectral representations for CNNs, which indeed allows expediting their training by 2-5 times: see Chapter 5.

Pooling layers

Pooling layers project onto a particular basis, and truncate the resultant representation. The goal of this operation is to perform dimensionality reduction, as well as imbue some desirable property such as translation invariance or reduced noise into the resultant output. The pooling itself is often performed across spatial components, and typically does not offer any parameters to be learned.

MAX POOLING Max pooling summarizes the contents of spatial windows via their maximal elements. More specifically, for a kernel of size $P \times Q$ and stride S, this is given by

$$\mathbf{f}(\mathbf{x})_{chw} = \max_{1 \le p \le P, 1 \le q \le Q} x_{c,\, hS+p,\, wS+q} \,. \tag{2.6}$$

Max pooling allows introducing translational invariance, since the maximum element in a particular window is invariant to its location within it. However, this property comes at the sacrifice of loss of information.

AVERAGE POOLING Average pooling is identical to max pooling, apart from the fact that the reduction operator is an average rather than a maximum. This form of pooling is linear, and in fact can be trivially reduced to a simple convolution with a constant kernel.

We remark that in this thesis, we introduce spectral pooling, which performs dimensionality reduction in the frequency domain. This approach allows preserving considerably more information per parameter than other pooling strategies, and enables flexibility in the choice of pooling output dimensionality. We expand on this in Chapter 5.

Nonlinearities

Nonlinearities allow us to introduce sparsity, contract to particular output ranges, and inject complexity. Without interlacing these between linear transformations, the entire network could be reduced to a single linear layer. Nonlinearities typically have no learnable parameters and are applied element-wise given an input tensor.

SIGMOID A sigmoidal nonlinearity is often denoted by $\sigma(\cdot)$, and corresponds to the transformation $\sigma(x) := 1/(1+e^{-x}) \in (0,1)$.

HYPERBOLIC TANGENT (TANH) As its name suggests, this nonlinearity is given by $\text{TanH}(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}} \in (-1, 1)$.

RECTIFIED LINEAR UNIT (RELU) The Rectified linear unit (ReLU) (Jarrett et al., 2009; Glorot et al., 2011b) is another name for the hinge function $\text{ReLU}(x) := \max(0, x)$. This nonlinearity allows achieving sparsity "easily" without requiring large domain values, and its piece-wise linear structure implies it does not saturate like bounded nonlinearities. The latter property allows it to better handle the vanishing gradient problem.

Stochastic regularization layers

DROPOUT Dropout has been introduced in Hinton et al. (2012b) to prevent overfitting resulting from co-adaptation of units. In its original formulation, at every iteration of training, a Bernoulli mask is sampled for each unit in the network, determining whether it is dropped from the network for that iteration. This implies that units cannot depend on the presence of others. During test-time, no dropout is applied, and the activations are halved to compensate for their joint presence.

NESTED DROPOUT In Chapter 3 of this thesis, we introduce Nested Dropout. This allows using a masking idea to attain representations that are inherently ordered by their importance. Such representations have a number of pleasing applications.

Losses

ℓ_p This class of losses is very popular in deep learning due to its ease of interpretation and implementation. For parameter p and $\mathbf{x} \in \mathbb{R}^K$, this loss is given by

$$\|\mathbf{x}\|_p = \left(\sum_{k=1}^{K} |x_k|^p \right)^{1/p} . \tag{2.7}$$

Average ℓ_2 loss is known as mean-square error (MSE), and is commonly used to penalize spatial reconstructions by measuring mean discrepancy per input, channel and pixel. Despite its ubiquitous adoption for this task, its use has been criticized, as it does not align well with the human perceptual system.

Another common loss in this family is ℓ_1, which is often employed as regularization to encourage sparsity.

SoftMax & Cross-Entropy The combination of SoftMax and cross-entropy is often used to as a loss to assess classification accuracy. Given a classification task with C classes, the inputs of this loss are a vector $\mathbf{r} \in \mathbb{R}^C$ produced by the network, and label $y \in \{1, \ldots, C\}$. The vector is then mapped to an array of probabilities of belonging to the different classes:

$$\mathbb{P}[y = c \,|\, \mathbf{r}] = \frac{e^{r_c}}{\sum_{c'=1}^{C} e^{r_{c'}}}, \qquad c = 1, \ldots, C\,. \tag{2.8}$$

This implies $\mathbf{p} \geq 0$ and $\sum_{c=1}^{C} p_c = 1$. The final objective for the particular input is then given by the cross-entropy between the Kronecker delta ground truth distribution and the computed probability:

$$\ell(\mathbf{r}, y) = -\sum_{c=1}^{C} \mathbb{1}_{y=c} \log \mathbb{P}[y = c \,|\, \mathbf{r}]\,. \tag{2.9}$$

Note that this combination of SoftMax and cross-entropy is often colloquially referred to as SoftMax, but this is technically incorrect: this term only describes the transformation in Equation (2.8).

2.3 Objective Optimization

Having fixed an architecture and objective for our problem, we proceed to optimize our loss from Equation (2.3):

$$\boldsymbol{\Theta}_1^\star, \ldots, \boldsymbol{\Theta}_M^\star = \arg \min_{\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_M} \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{r}_n^{(M)}, y_n)\,. \tag{2.10}$$

This is a notoriously difficult optimization problem: the objective is non-convex and non-stationary, and its minimization is commonly performed over hundreds of millions of parameters. Moreover, for very large datasets, only a tiny fraction of the inputs can in fact fit in memory, and as such only stochastic approaches are computationally feasible.

In practice, this optimization is performed iteratively via gradient-based approaches. The gradient can be computed efficiently using *back-propagation* (Rumelhart et al., 1986), which refers to computation of the chain rule via reverse mode differentiation.

Stochastic optimization

Oftentimes, larger datasets cannot fit in memory, and this prohibits direct computation of the objective $\frac{1}{N}\sum_{n=1}^{N}\ell(\mathbf{r}_n^{(M)}, y_n)$. This necessitates *stochastic optimization*, in which the global objective is approximated via *minibatches* of examples sampled at each iteration of optimization:

$$\begin{aligned} \frac{1}{N}\sum_{n=1}^{N}\ell(\mathbf{r}_n^{(M)}, y_n) &\approx \frac{1}{|\mathcal{B}|}\sum_{(\mathbf{x},y)\in\mathcal{B}}\ell(\mathbf{r}^{(M)}, y)\,, \qquad \mathcal{B}\subset\mathcal{D},\ |\mathcal{B}|\ll|\mathcal{D}| \qquad (2.11)\\ &=: C_{\mathcal{B}}(\mathbf{\Theta}) \qquad (2.12)\end{aligned}$$

for model parameters $\mathbf{\Theta}$. Minibatch size provides a tradeoff between computational expense and accurate approximation of the objective. While the cost of gradient computation is linear as function of minibatch size, the returns in terms of expedited convergence diminish. Typical minibatch size is on the order of magnitude of tens or hundreds of examples. This loss, and its gradient, could be computed by accumulating results over different minibatches. However, estimating the objective with individual minibatches and applying respective updates is significantly more efficient. Minibatches are often sampled uniformly from the training set; however, there exists work on *curriculum learning*, where examples are presented to the optimizer in a particular sequence, ordered by metrics such as difficulty (Bengio et al., 2009).

Common gradient-based optimizers

Gradient-based optimizers correspond to different update rules for adjusting the model parameters $\mathbf{\Theta}$ given the gradient $\nabla_{\mathbf{\Theta}} C_{\mathcal{B}}(\mathbf{\Theta})$ and possibly second-order information at the current location.

There exist many optimizers, motivated by exploitations of different assumptions with respect to the structure of the objective in order to facilitate convergence. While these optimizers are often backed by theoretical guarantees, these are often proven under very constraining assumptions. However, these methods are then deployed in the wild in settings where the premises do not hold: as such, their respective advantages are often subdued in real-life optimization circumstances. Hence, the evaluation of different optimization strategies often reduces to empirical assessment across different objectives and hyperparameter configurations.

Irrespective of the optimizer, *weight decay* is often applied to the non-bias weights at each iteration as element-wise rescaling $\mathbf{\Theta}_t = (1-\varepsilon)\mathbf{\Theta}_{t-1}$, $0 < \varepsilon \ll 1$. This corresponds to an ℓ_2 regularization objective on the weights. This wipes out irrelevant coefficients, and as such acts as a form of regularization.

Stochastic Gradient Descent (SGD) This is the simplest optimizer. It is based on the following update rule for iteration t:

$$\mathbf{\Theta}_t = \mathbf{\Theta}_{t-1} - \alpha_t \nabla_{\mathbf{\Theta}} C_{\mathcal{B}}(\mathbf{\Theta}_{t-1})\,. \tag{2.13}$$

Here, $\alpha_t > 0$ is the *learning rate.* The learning rate determines the weight of the current update. For this optimizer, it varies as function of objective lengthscale, and is often annealed over time to produce gentler updates as the procedure nears convergence.

SGD WITH MOMENTUM This is one of the most common optimizers in deep learning. It is based on the following update rule for iteration t:

$$\boldsymbol{\mu}_t = \nu\boldsymbol{\mu}_{t-1} - \alpha_t \nabla_{\boldsymbol{\Theta}} C_{\mathcal{B}}(\boldsymbol{\Theta}_{t-1}) \,, \tag{2.14}$$

$$\boldsymbol{\Theta}_t = \boldsymbol{\Theta}_{t-1} + \boldsymbol{\mu}_t \,. \tag{2.15}$$

Here, $\nu \in (0,1)$ is the *momentum*. For momenutum $\nu = 0$, this reduces exactly to the SGD update rule in Equation (2.13). The learning rate determines the weight of the current update. The momentum enables accruing gradients across iterations, which serves as a form of variance reduction: it ameliorates stability, which is often harmed due to the stochastic objective approximation. In practice, momentum is canonically taken as $\nu \approx 0.9$.

ADAM This optimizer was introduced in Kingma & Ba (2015), where its implementation details can be found. It is a robust extension of AdaGrad (Duchi et al., 2011). It is a first-order method invariant to diagonal rescaling of the gradients. It enables element-wise rescaling of the learning rate, which significantly expedites convergence and counters the vanishing gradient pathology exhibited by SGD and similar optimizers. In our experiments, we found Adam to be a versatile optimizer, suitable for model prototyping: it is able to optimize a range of problems with a fixed out-of-the-box hyperparameter configuration.

Chapter 3

Learning Ordered Representations with Nested Dropout

3.1 Introduction

One frustration associated with current representation learning techniques is redundancy from non-identifiability in the resulting encoder/decoder. That is, under standard models such as autoencoders, restricted Boltzmann machines, and sparse coding, any given solution is part of an *equivalence class* of solutions that are equally optimal. This class emerges from the invariance of the models to various transformations of the parameters. Permutation is one clear example of such parameter transformation, leading to a combinatorial number of equivalent representations for a given dataset and architecture. There exist many other types of redundancies as well; the optimality of an autoencoder solution is preserved under any invertible linear transformation of the innermost set of weights (Bourlard & Kamp, 1987). This degeneracy also poses a difficulty when comparing experiments, due to the lack of repeatability: a solution attained by the optimization procedure is extremely sensitive to the choice of initialization.

This large number of equivalent representations has an advantage, however: it provides flexibility in architecture design. This freedom allows us to impose desirable structural constraints on the learned representations, without compromising their expressiveness. These constraints can imbue a number of useful properties, including the elimination of permutation non-identifiability. In this work we propose one such structural constraint: we specify *a priori* the quantity of information encapsulated in each dimension of the representation. This choice allows us to order the representation dimensions according to their information content.

The intuition behind our proposed approach to learning ordered representations is to train models such that the information contained in each dimension of the representation decreases as a function of the dimension index, following a pre-specified decay function. To this end, we introduce the *nested dropout* algorithm. As with the original dropout formulation (Hinton et al., 2012b), nested dropout applies a stochastic mask over models. However, instead of imposing an independent distribution over each individual unit in a model, it assigns a distribution over nested subsets of representation units. More specifically, given a representation space of dimension K, we define a distribution $p_B(\cdot)$ over the representation index subsets $S_b = \{1, \ldots, b\}$, $b = 1, \ldots, K$. This has the property that if the j-th unit appears in a particular mask, then so do all "earlier" units $1, \ldots, j-1$, allowing the j-th unit to depend on their values. This nesting leads to an inherent ordering over the representation dimensions. The distribution $p_B(\cdot)$ then governs the information capacity decay by modulating the relative frequencies of these masks. We motivate such ordered representations in several ways, described below.

IDENTIFIABILITY As discussed above, many current representation learning techniques suffer from non-identifiability of the solutions. We can remedy this by introducing strict representation ordering, which enforces distinguishability. We rigorously demonstrate this for the special case of a semi-linear autoencoder. We prove that the application of nested dropout leads to a significant reduction in the solution space complexity without harming the solution quality. Under an additional weak constraint, we further prove that, modulo sign flips, the model has a single and unique global optimum. We show that this solution is exactly the set of eigenvectors of the covariance matrix of the data, ordered by eigenvalue magnitude. This demonstrates exact equivalence between semi-linear nested dropout autoencoders and principal component analysis (PCA).

FAST RETRIEVAL Current information retrieval procedures suffer from an intrinsic tradeoff between search speed and quality: representation dimensionality and dataset size must be sacrificed to gain search tractability (Grauman & Fergus (2013) offers an excellent overview of modern retrieval procedures). Given a query datum, a naïve brute force retrieval based on Hamming distance requires a linear scan of the database, which has complexity $\mathcal{O}(KN)$ where K is the code length and N the database size. Semantic hashing (Salakhutdinov & Hinton, 2009) retrieves examples within a Hamming neighborhood of radius R by directly scanning through all memory locations associated with them. This results in retrieval time complexity $\mathcal{O}(\binom{K}{R})$. While this is independent of the database size, it grows rapidly in K and therefore is computationally prohibitive even for codes tens of bits long; code length of 50 bits, for example, requires a petabyte of memory be addressed. Moreover, as the code length increases, it becomes very likely that many queries will not find any neighbors for any feasible radii. Locality sensitive hashing (Datar et al., 2004) seeks to preserve distance information by means of random projections; however, this can lead to very inefficient codes for high input dimensionality.

By imposing an ordering on the information represented in a deep model, we can learn hash functions that permit efficient retrieval. Because the importance of each successive coding dimension decays as we move through the ordering, we can naturally construct a binary tree data structure on the representation to capture a coarse-to-fine notion of similarity. This allows retrieval in time that is logarithmic with the dataset size and *independent* of the representation space dimensionality. This enables very fast retrieval on large databases without sacrificing representation quality: we are able to consider codes hundreds of times longer than currently feasible with existing retrieval methods. For example, we perform retrieval on a dataset of a million entries of code length 2048 in an average time of $200\mu s$ per query—about 4 orders of magnitude faster than a linear scan or semantic hashing.

ADAPTIVE COMPRESSION Ordered representations can also be used for "continuous-degradation" lossy compression systems: they give rise to a continuous range of bitrate/quality combinations, where each additional bit corresponds to a small incremental increase in quality. This property can in principle be applied to problems such as video streaming. The representation only needs to be encoded a single time; then, users of different bandwidths can be adaptively sent codes of different length that exactly match their bitrates. The inputs can then be reconstructed optimally for the users' channel capacities.

3.2 Ordering with Nested Dropout

Dropout (Hinton et al., 2012b) is a regularization technique for neural networks that adds stochasticity to the architecture during training. At each iteration, Bernoulli coins are flipped independently for each unit in the network, determining whether it is "dropped" or not. Every dropped unit is deleted from the network for that iteration, and an optimization step is taken with respect to the resulting network.

Nested dropout diverges from this in two main ways. First, only representation units are dropped. Second, instead of flipping independent coins for different units, we instead assign a distribution $p_B(\cdot)$ over the representation *indices* $1, \ldots, K$. We then sample an index $b \sim p_B(\cdot)$ and drop units $b+1, \ldots, K$. The sampled units then form nested subsets: if unit j appears in a network sample, then so do units $1, \ldots, j-1$. This nesting results in an inherent importance ranking of the representation dimensions, as a particular unit can always rely on the presence of its predecessors. For $p_B(\cdot)$ we select a geometric distribution: $p_B(b) = \rho^{b-1}(1-\rho)$. We make this choice due to the exponential decay of this distribution and its memoryless property (see Section 3.4).

We construct our model as an autoencoder, which is a parametric composition of an encoder and a decoder. We are given a set of N training examples $\{\mathbf{y}_n\}_{n=1}^{N}$ lying in space $\mathscr{Y} \subseteq \mathbb{R}^D$. We then transform the data into the *representation space* $\mathscr{X} \subseteq \mathbb{R}^K$ via a parametric transformation $\boldsymbol{f}_{\Theta} : \mathscr{Y} \to \mathscr{X}$. We denote this function as the *en-*

coder, and label the representations as $\{\mathbf{x}_n\}_{n=1}^N \subset \mathscr{X}$. The *decoder* map $\boldsymbol{g}_{\boldsymbol{\Psi}} : \mathscr{X} \to \mathscr{Y}$ then reconstructs the inputs from their representations as $\{\hat{\mathbf{y}}_n\}_{n=1}^N$.

A SINGLE NESTED DROPOUT SAMPLE Let us assume that we sample some $b \sim p_B(\cdot)$ and drop the last $K - b$ representation units; we refer to this case as the *b-truncation*. This structure is equivalent to an autoencoder with a representation layer of dimension b. For a given representation $\mathbf{x} \in \mathbb{R}^K$, we define $\mathbf{x}_{\downarrow b}$ as the truncation of the vector $\mathbf{x}$ where the last $K - b$ elements are removed.

Denoting the reconstruction of the b-truncation as $\hat{\mathbf{y}}_{\downarrow b} = \boldsymbol{g}_{\boldsymbol{\Psi}}(\boldsymbol{f}_{\boldsymbol{\Theta}}(\mathbf{y})_{\downarrow b})$, the reconstruction cost function associated with a b-truncation is then

$$C_{\downarrow b}(\boldsymbol{\Theta}, \boldsymbol{\Psi}) = \frac{1}{N} \sum_{n=1}^{N} \ell\left(\mathbf{y}_n, \hat{\mathbf{y}}_{n\downarrow b}\right) \,. \tag{3.1}$$

In this work, we take the reconstruction loss $\ell(\cdot, \cdot)$ to be the ℓ_2 norm. Although we write this cost as a function of the full parametrization $(\boldsymbol{\Theta}, \boldsymbol{\Psi})$, due to the truncation only a subset of the parameters will contribute to the objective.

THE NESTED DROPOUT PROBLEM Given our distribution $p_B(\cdot)$, we consider the mixture of the different b-truncation objectives:

$$C(\boldsymbol{\Theta}, \boldsymbol{\Psi}) = \mathbb{E}_B\left[C_{\downarrow b}(\boldsymbol{\Theta}, \boldsymbol{\Psi})\right] = \sum_{b=1}^{K} p_B(b) C_{\downarrow b}(\boldsymbol{\Theta}, \boldsymbol{\Psi}). \tag{3.2}$$

We formulate the *nested dropout* problem as the optimization of this mixture with respect to the model parameters:

$$(\boldsymbol{\Theta}^*, \boldsymbol{\Psi}^*) = \arg\min_{\boldsymbol{\Theta}, \boldsymbol{\Psi}} \; C(\boldsymbol{\Theta}, \boldsymbol{\Psi}) \,. \tag{3.3}$$

3.2.1 Interpretation

Nested dropout has a natural interpretation in terms of information content in representation units. It was shown by Vincent et al. (2010) that training an autoencoder corresponds to maximizing a lower bound on the mutual information $\mathcal{I}(\boldsymbol{y};\boldsymbol{x})$ between the input data and their representations. Specifically, the objective of the b-truncation problem can be written in the form

$$C_{\downarrow b}(\boldsymbol{\Theta},\boldsymbol{\Psi}) \approx \mathbb{E}_{\mathbf{y}}\left[-\log p_{\boldsymbol{Y}|\boldsymbol{X}_{\downarrow b}}(\mathbf{y}\,|\,\boldsymbol{f}_{\boldsymbol{\Theta}}(\mathbf{y})_{\downarrow b}\,;\,\boldsymbol{\Psi})\right] \tag{3.4}$$

where we assume our data are sampled from the true distribution $p_{\boldsymbol{Y}}(\cdot)$. The choice $p_{\boldsymbol{Y}|\boldsymbol{X}}(\mathbf{y}\,|\,\mathbf{x}\,;\,\boldsymbol{\Psi}) = \mathcal{N}(\mathbf{y}\,;\,\boldsymbol{g}_{\boldsymbol{\Psi}}(\mathbf{x}),\sigma^2\mathbb{I}_D)$, for example, leads to the familiar autoencoder ℓ_2 reconstruction penalty.

Now, define $\tilde{\mathcal{I}}_b(\boldsymbol{y};\boldsymbol{x}) := -C_{\downarrow b}(\boldsymbol{\Theta},\boldsymbol{\Psi}) \leq \mathcal{I}(\boldsymbol{y};\boldsymbol{x})$ as the approximation of the true mutual information which we maximize for a given b. Then we can write the (negative) nested dropout problem in the form of a telescopic sum:

$$\begin{aligned} -C(\boldsymbol{\Theta},\boldsymbol{\Psi}) &= \sum_{b=1}^{K} p_B(b)\tilde{\mathcal{I}}_b(\boldsymbol{y};\boldsymbol{x}) \\ &= \tilde{\mathcal{I}}_1(\boldsymbol{y};\boldsymbol{x}) + \sum_{b=2}^{K}\left[F_B(K) - F_B(b-1)\right]\Delta_b\,, \end{aligned} \tag{3.5}$$

where $F_B(b) = \sum_{b'=1}^{b} p_B(b')$ is the cumulative distribution function of $p_B(\cdot)$, and $\Delta_b := \tilde{\mathcal{I}}_b(\boldsymbol{y};\boldsymbol{x}) - \tilde{\mathcal{I}}_{b-1}(\boldsymbol{y};\boldsymbol{x})$ is the marginal information gained from increasing the representation dimensionality from b units to $b+1$.

This formulation provides a connection between the nested dropout objective and the optimal distribution of information across the representation dimensions. Note that the coefficients $F_B(K) - F_B(b)$ of the marginal mutual information are positive and monotonically decrease as a function of b regardless of the choice of distribution $p_B(\cdot)$. This establishes the ordering property intuitively sought by the nested dropout idea. We also see that if for some b we have $p_B(b) = 0$, i.e., index b has no support under $p_B(\cdot)$, then the ordering of representation dimensions b and $b-1$ no longer matters. If we set $p_B(1) = 0, \ldots, p_B(K-1) = 0$ and $p_B(K) = 1$, we recover the original order-free autoencoder formulation for K latent dimensions. In order to achieve strict ordering, then, the only assumption we must make is that $p_B(\cdot)$ has support over all representation indices. Indeed, this will be a sufficient condition for our proofs in Section 3.3. Equation (3.5) informs us of how our prior choice of $p_B(\cdot)$ dictates the optimal information allocation per unit.

3.3 Exact Recovery of PCA

In this section, we apply nested dropout to a semi-linear autoencoder. This model has a linear or a sigmoidal encoder, and a linear decoder. The relative simplicity of this case allows us to rigorously study the ordering property implied by nested dropout.

First, we show that the class of optimal solutions of the nested dropout autoencoder is a subset of the class of optimal solutions of a standard autoencoder. This means that introducing nested dropout does not sacrifice the quality of the autoencoder solution. Second, we show that equipping an autoencoder with nested dropout significantly constrains its class of optimal solutions. We characterize these restrictions. Last, we show that under an additional orthonormality constraint, the model features a single, unique solution that is exactly the set of K eigenvectors with the largest magnitudes arising from the covariance matrix of the inputs, ordered by decreasing eigenvalue magnitude. Hence this recovers the PCA solution exactly. This is in contrast to a standard autoencoder, which recovers the PCA solution up to an invertible linear map.

3.3.1 Problem definitions and prior results

THE STANDARD LINEAR AUTOENCODER PROBLEM Given our inputs, we apply the linear encoder $\boldsymbol{f}_{\boldsymbol{\Theta}}(\mathbf{y}) := \boldsymbol{\Omega}\mathbf{y} + \boldsymbol{\omega}$ with parameters $\boldsymbol{\Omega} \in \mathbb{R}^{K \times D}$ and bias vector $\boldsymbol{\omega} \in \mathbb{R}^K$ for $K \leq D$. Our proofs further generalize to sigmoidal nonlinearities applied to the output of the encoder, but we omit these for clarity. The decoder map $\boldsymbol{g}_{\boldsymbol{\Psi}} : \mathscr{X} \to \mathscr{Y}$ is similarly taken to be $\boldsymbol{g}_{\boldsymbol{\Psi}}(\mathbf{x}) := \boldsymbol{\Gamma}\mathbf{x} + \boldsymbol{\gamma}$ with parameters $\boldsymbol{\Gamma} \in \mathbb{R}^{D \times K}$ and $\boldsymbol{\gamma} \in \mathbb{R}^D$. We also define the design matrices $\boldsymbol{Y}$ and $\boldsymbol{X}$ whose columns consist of the observations and their representations, respectively.

The reconstruction of each datum is then defined as the composition of the encoder and decoder maps. Namely, $\hat{\mathbf{y}}_n = \mathbf{\Gamma}(\mathbf{\Omega}\mathbf{y}_n + \boldsymbol{\omega}) + \boldsymbol{\gamma}\ \forall n = 1, \ldots, N$. A semi-linear autoencoder seeks to minimize the reconstruction cost

$$\begin{aligned} C(\mathbf{\Theta}, \mathbf{\Psi}) &= \sum_{n=1}^{N} \|\mathbf{y}_n - \boldsymbol{g}_{\mathbf{\Psi}}(\boldsymbol{f}_{\mathbf{\Theta}}(\mathbf{y}_n))\|^2 && (3.6) \\ &= \|\boldsymbol{Y} - (\mathbf{\Gamma}(\mathbf{\Omega}\boldsymbol{Y} + \boldsymbol{\omega}) + \boldsymbol{\gamma})\|_F^2 && (3.7) \end{aligned}$$

where by $\|\cdot\|_F$ we denote the Frobenius matrix norm. From this point on, without loss of generality we assume that $\boldsymbol{\omega} = \mathbf{0}$, $\boldsymbol{\gamma} = \mathbf{0}$, and that the data are zero-centered. All our results hold otherwise, but with added shifting constants.

A SINGLE b-TRUNCATION PROBLEM We continue to consider the b-truncation problem, where the last $K - b$ units of the representation are dropped. As before, for a given representation $\mathbf{x} \in \mathbb{R}^K$, we define $\mathbf{x}_{\downarrow b}$ to be the truncation of vector $\mathbf{x}$. Defining the truncation matrix $\boldsymbol{J}_{m \to n} \in \mathbb{R}^{n \times m}$ as $[\boldsymbol{J}_{m \to n}]_{ab} = \delta_{ab}$, then $\mathbf{x}_{\downarrow b} = \boldsymbol{J}_{K \to b}\mathbf{x}$. The decoder is then written as $\boldsymbol{g}_{\mathbf{\Psi}\downarrow b}(\mathbf{x}_{\downarrow b}) = \mathbf{\Gamma}_{\downarrow b}\mathbf{x}_{\downarrow b}$, where we write $\mathbf{\Gamma}_{\downarrow b} = \mathbf{\Gamma}\boldsymbol{J}_{K \to b}^T$ in which the last $K - b$ columns of $\mathbf{\Gamma}$ are removed. The reconstruction cost function associated with a b-truncation is then

$$C_{\downarrow b}(\mathbf{\Theta}_{\downarrow b}, \mathbf{\Psi}_{\downarrow b}) = \|\boldsymbol{Y} - \Gamma_{\downarrow b}\boldsymbol{X}_{\downarrow b}\|_F^2 \ . \tag{3.8}$$

We define $(\mathbf{\Theta}_{\downarrow b}^*, \mathbf{\Psi}_{\downarrow b}^*) = \arg\min_{\mathbf{\Theta}_{\downarrow b}, \mathbf{\Psi}_{\downarrow b}} C_{\downarrow b}(\mathbf{\Theta}_{\downarrow b}, \mathbf{\Psi}_{\downarrow b})$ to be an optimal solution of the b-truncation problem; we label the corresponding optimal cost as $C_{\downarrow b}^*$. Also, let $\boldsymbol{V}_{\boldsymbol{Y}} = \boldsymbol{Y}\boldsymbol{Y}^T$ be (proportional to) the empirical covariance matrix of $\{\mathbf{y}_n\}_{n=1}^N$ with eigendecomposition $\boldsymbol{V}_{\boldsymbol{Y}} = \boldsymbol{Q}\Sigma^2\boldsymbol{Q}^T$, where Σ^2 is the diagonal matrix constituting of the eigenvalues arranged in decreasing magnitude order, and $\boldsymbol{Q}$ the orthonormal matrix of the respective eigenvectors. Similarly, let $\boldsymbol{R}$ be the orthonormal eigenvector matrix of $\boldsymbol{Y}^T\boldsymbol{Y}$, arranged by decreasing order of eigenvalue magnitude.

The b-truncation problem exactly corresponds to the original semi-linear autoencoder problem, where the representation dimension is taken to be b in the first place. As such, we can apply known results about the form of the solution of a standard autoencoder. It was proven in Bourlard & Kamp (1987) that this optimal solution must be of the form

$$\boldsymbol{X}_b^* = \boldsymbol{T}_b \boldsymbol{\Sigma}_{\downarrow b} \boldsymbol{R}^T \qquad\qquad \boldsymbol{\Gamma}_b^* = \boldsymbol{Q}_{\downarrow b} \boldsymbol{T}_b^{-1} \tag{3.9}$$

where $\boldsymbol{T}_b \in \mathbb{R}^{b \times b}$ is an invertible matrix, $\boldsymbol{\Sigma}_{\downarrow b} = \boldsymbol{J}_{K \to b} \boldsymbol{\Sigma} \in \mathbb{R}^{b \times D}$ the matrix with the b largest-magnitude eigenvalues, and $\boldsymbol{Q}_{\downarrow b} = \boldsymbol{Q} \boldsymbol{J}_{K \to b}^T \in \mathbb{R}^{D \times b}$ the matrix with the b corresponding eigenvectors. This result was established for an autoencoder of representation dimension b; we reformulated the notation to suit the nested dropout problem we define in the next subsection.

It can be observed from Equation (3.9) that the semi-linear autoencoder has a strong connection to PCA. An autoencoder discovers the eigenvectors of the empirical covariance matrix of $\{\mathbf{y}_n\}_{n=1}^N$ corresponding to its b eigenvalues of greatest magnitude; however, this is up to to an invertible linear transformation. This class includes rotations, scalings, reflections, index permutations, and so on. This non-identifiability has an undesirable consequence: it begets a huge class of optimal solutions.

THE NESTED DROPOUT PROBLEM We now introduce the nested dropout problem. Here, we assign the distribution $b \sim p_B(\cdot)$ as a prior over b-truncations. For our proofs to hold our only assumption about this distribution is that it has support over the entire index set, i.e., $p_B(b) > 0,\ \forall b = 1, \ldots, K$. To that end, we seek to minimize the nested dropout cost function, which we define as the mixture of the K truncated models under $p_B(\cdot)$:

$$C(\boldsymbol{\Theta}, \boldsymbol{\Psi}) = \mathbb{E}_B \left[\|\boldsymbol{Y} - \Gamma_{\downarrow b} \boldsymbol{X}_{\downarrow b}\|_F^2 \right] \tag{3.10}$$

$$= \sum_{b=1}^{K} p_B(b) \left\| \boldsymbol{Y} - \Gamma_{\downarrow b} \boldsymbol{X}_{\downarrow b} \right\|_F^2 . \tag{3.11}$$

3.3.2 The nested dropout problem recovers PCA exactly

Below we provide theoretical justification for the claims made in the beginning of this section. All of the proofs can be found in Appendix A.

Theorem 1. *Every optimal solution of the nested dropout problem is necessarily an optimal solution of the standard autoencoder problem.*

Definition. *We define matrix $\boldsymbol{T} \in \mathbb{R}^{K\times K}$ to be* commutative in its truncation and inversion *if each of its leading principal minors $\boldsymbol{J}_{K\to b}\boldsymbol{T}\boldsymbol{J}_{K\to b}^T$, $b = 1, \dots, K$ is invertible, and the inverse of each of its leading principal minors is equal to the leading principal minor of the inverse $\boldsymbol{T}^{-1}$, namely*

$$\boldsymbol{J}_{K\to b}\boldsymbol{T}^{-1}\boldsymbol{J}_{K\to b}^T = (\boldsymbol{J}_{K\to b}\boldsymbol{T}\boldsymbol{J}_{K\to b}^T)^{-1} . \tag{3.12}$$

The below theorem, combined with Lemma 1, establishes tight constraints on the class of optimal solutions of the nested dropout problem. For example, an immediate corollary of this is that $\boldsymbol{T}$ cannot be a permutation matrix, as for such a matrix there must exist some leading principal minor that is not invertible.

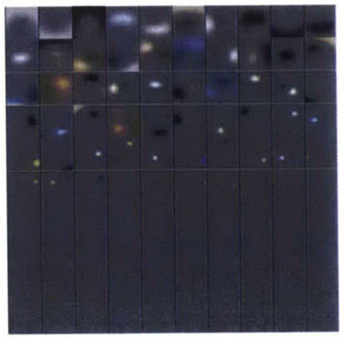
Theorem 2. *Every optimal solution of the nested dropout problem must be of the form*

$$\boldsymbol{X}^* = \boldsymbol{T}\boldsymbol{\Sigma}\boldsymbol{R}^T \qquad\qquad \boldsymbol{\Gamma}^* = \boldsymbol{Q}\boldsymbol{T}^{-1} , \tag{3.13}$$

for some matrix $\boldsymbol{T} \in \mathbb{R}^{K\times K}$ that is commutative in its truncation and inversion.

Denote the column and row submatrices respectively as $\boldsymbol{A}_b = [T_{1b}, \dots, T_{(b-1),b}]$ and $\boldsymbol{B}_b = [T_{b1}, \dots, T_{b,(b-1)}]^T$.

Lemma 1. *Let $\boldsymbol{T} \in \mathbb{R}^{K\times K}$ be commutative in its truncation and inversion. Then all the diagonal elements of $\boldsymbol{T}$ are nonzero, and for each $b = 2, \dots, K$, either $\boldsymbol{A}_b = \boldsymbol{0}$ or $\boldsymbol{B}_b = \boldsymbol{0}$.*


(a) Without unit sweeping

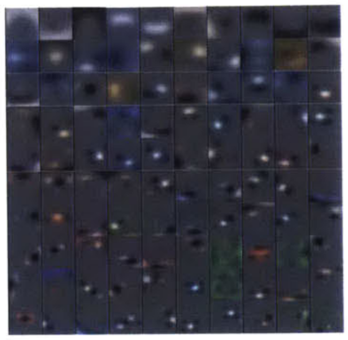

(b) With unit sweeping

Figure 3-1: The 100 filters learned by a binarized 3072-100-3072 nested dropout autoencoder on the raw CIFAR-10 pixels where $p_B(\cdot)$ is a geometric distribution with rate of 0.9. For this rate, the probability of sampling any index greater than 50 is ≈ 0.005, and the probability of sampling the 100th unit is ≈ 0.00003: it is very unlikely to ever sample these without unit sweeping. Note the increase of filter fineness as a function of the index.

In the result below we see that nested dropout coupled with an orthonormality constraint effectively eliminates non-identifiability. The added constraint pins down any possible rotations and scalings.

Theorem 3. *Under the orthonormality constraint $\mathbf{\Gamma}^T\mathbf{\Gamma} = \mathbb{I}_K$, the nested dropout problem features a unique global optimum, and this solution is exactly the set of the K top eigenvectors of the covariance of $\boldsymbol{Y}$, ordered by eigenvalue magnitude. Namely, $\boldsymbol{X}^* = \mathbf{\Sigma}\boldsymbol{R}^T, \mathbf{\Gamma}^* = \boldsymbol{Q}$.*

3.4 Training Deep Models with Nested Dropout

In this section we discuss our extension of the nested dropout approach to deep architectures. Specifically, we applied this to deep autoencoders having tens of millions of parameters, which we trained on the 80 Million Tiny Images (80MTI) dataset (Torralba et al., 2008) on two GPUs. Training models with nested dropout introduces a number of unconventional technical challenges. In the following sections we describe these challenges, and present strategies to overcome them.

We first describe our general architecture and optimization setup. The 80MTI are

79,302,017 color images of size 32×32. We pre-processed the data by subtracting from each pixel its mean and normalizing by its variance across the dataset. We optimize our models with the nonlinear conjugate gradients algorithm and select step sizes using a strong Wolfe conditions line search. For retrieval-related tasks, we seek to produce binary representations. In light of this we use rectified linear units for all nonlinearities in our encoder, as we find this leads to better binarized representation (see Subsection **3.4.3**). We train for 2 epochs on minibatches of size 10,000. We inject noise to promote robustness, as in (**Vincent et al., 2010**); namely, with probability 0.1 we independently corrupt input elements to 0. For all layers other than the representation layer, we apply standard dropout with probability 0.2. At each iteration, we sample nested dropout truncation indices for each example in our minibatch, and take a step with respect to the corresponding network mask.

3.4.1 Unit sweeping for decaying gradients

By the virtue of the decaying distribution $p_B(\cdot)$, it becomes increasingly improbable to sample higher representation indices during training. As such, we encounter a phenomenon where gradient magnitudes vanish as a function of representation unit index. This curvature pathology, in its raw formulation, means that training representation units of higher index can be extremely slow.

In order to combat this effect, we develop a technique we call *unit sweeping*. The idea stems from the observation that the covariance of two latent units sharply decreases as a function of the of the difference of their indices. When $p_B(\cdot)$ is a geometric distribution, for example, the probability of observing both units i and j given that one of them is observed is $\mathbb{P}[b \geq \max(i,j) \mid b \geq \min(i,j)] = \mathbb{P}[b \geq |i-j|] = \rho^{-|i-j|}$ by the memoryless property of the distribution. In other words, a particular latent unit becomes exponentially desensitized to values of units of higher index. As such, this unit will eventually converge during its training. Upon convergence, then, this unit can be fixed in place and its associated gradients can be omitted. Loosely speaking, this elimination reduces the "condition number" of the optimization. Applying this iteratively, we sweep through the latent units, fixing each once it converges. In Figure

3-1 we compare filters from training a nested dropout model with and without unit sweeping.

3.4.2 Adaptive regularization coefficients

The gradient decay as a function of representation index poses a difficulty for regularization. In particular, the ratio of the magnitudes of the gradients of the reconstruction and the regularization vanishes as a function of the index. Therefore, a single regularization term such as $\lambda \sum_{k=1}^{K} \|\mathbf{\Omega}_k\|_1$ on the weights of the layer mapping to our representations would not be appropriate for nested dropout, since the regularization gradient would dominate the high-index gradients. As such, the regularization strength must be a function of representation index. For weight decay, for example, this would of the form $\sum_{k=1}^{K} \lambda_k \|\mathbf{\Omega}_k\|_1$. Choosing the coefficients λ_k manually is challenging, and to that end we assign them adaptively. We do this by fixing in advance the ratio between the magnitude of the reconstruction gradient and the regularization gradient, and choosing the λ_k to satisfy this ratio requirement. This corresponds to fixing the relative contributions of the objective and regularizations terms to the gradient for each step of the optimization procedure.

3.4.3 Code binarization

For the task of retrieval, we would like to obtain binary representations. Several binarization methods have been proposed in prior work (Salakhutdinov & Hinton, 2009; Krizhevsky & Hinton, 2011). We have empirically achieved good performance by tying the weights of the encoder and decoder, and thresholding at the representation layer. Although the gradient itself cannot propagate past this threshold, some signal does: the encoder can be trained since it is linked to the decoder, and its modifications are then reflected in the objective. To attain fixed marginal distributions over the binarized representation units, i.e., $x_k \sim \text{Bern}(\beta)$ for $k = 1, \ldots, K$, we compute the β quantile for each unit, and use this value for thresholding.

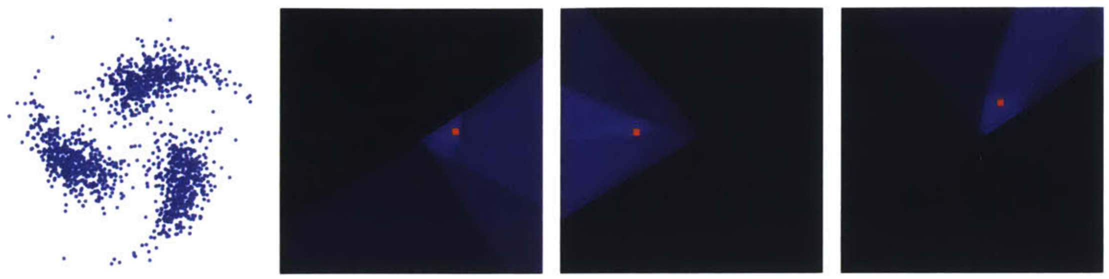


Figure 3-2: Nested neighborhoods for various examples in a 2D toy problem. The synthetic pinwheel training data is on the top left. The trained model is a 2-16-32-16-2 nested dropout autoencoder. The red dots correspond to the retrieved queries. The shades of blue correspond to different nested neighborhoods for these queries, with color lightness signifying neighborhood depth.

3.5 Retrieval With Ordered Binary Codes

In this section we discuss how ordered representations can be exploited to construct data structures that permit fast retrieval while at the same time allowing for very long codes.

3.5.1 Binary tree on the representation space

The ordering property, coupled with the ability to control information capacity decay across representation units, motivates the construction of a binary tree over large data sets. Each node in this tree contains pointers to the set of examples that share the same path down the tree up to that point. Guaranteeing that this tree is balanced is not feasible, as this is equivalent to completely characterizing the joint distribution over the representation space. However, by the properties of the training algorithm, we are able to fix the marginal distributions of all the representation bits as $x_k \sim \text{Bern}(\beta)$ for some hyperparameter $\beta \in (0, 1)$.

Consistent with the training procedure, we encode our database as $\boldsymbol{X} = \{\mathbf{x}_n\}_{n=1}^{N} \subseteq \{0,1\}^K$ for $\mathbf{x}_n = \boldsymbol{f}_{\boldsymbol{\Theta}}(\mathbf{y}_n)$. We then construct a binary tree on the resulting codes.

Given a query $\bar{\mathbf{y}}$, we first encode it as $\bar{\mathbf{x}} = \boldsymbol{f}_{\boldsymbol{\Theta}}(\bar{\mathbf{y}}_n)$. We then conduct retrieval by traveling down the binary tree with each branching determined by the next bit of $\bar{\boldsymbol{x}}$. We define the *b-truncated Hamming neighborhood* of $\bar{\mathbf{x}}$ as the set of all examples whose codes share the first b bits of $\bar{\mathbf{x}}$:

$$N_b^{\mathcal{H}}(\bar{\mathbf{x}}) = \left\{\mathbf{y} \in \boldsymbol{Y} : \|\mathbf{x}_{\downarrow b} - \bar{\mathbf{x}}_{\downarrow b}\|_{\mathcal{H}} = 0\right\} . \tag{3.14}$$

It is clear that $N_{b+1}^{\mathcal{H}}(\bar{\mathbf{x}}) \subseteq N_b^{\mathcal{H}}(\bar{\mathbf{x}})\ \forall b = 1, \dots, K-1$. Our retrieval procedure then corresponds to iterating through this family of nested neighborhoods. We expect the cardinality of these to decay approximately exponentially as a function of index. We terminate the retrieval procedure when $\left|N_b^{\mathcal{H}}(\bar{\mathbf{x}})\right| < R$ for some pre-specified terminal neighborhood cardinality, $R \in \mathbb{N}$. It outputs the set $N_{b-1}^{\mathcal{H}}(\bar{\mathbf{x}})$.

Assuming marginals $x_k \sim \text{Bern}(\beta)$ and neglecting dependence between the x_k, this results in expected retrieval time $\mathcal{O}(\frac{\log N/R}{\mathcal{H}(\text{Bern}(\beta))})$ where $\mathcal{H}(\text{Bern}(\beta))$ is the Bernoulli entropy. If $\beta = \frac{1}{2}$, for example, this reduces to the balanced tree travel time $\mathcal{O}(\log N/R)$. This retrieval time is logarithmic in the database size N, and *independent* of the representation space dimensionality K. If one wishes to retrieve a fixed fraction of the dataset, this renders the retrieval complexity also independent of the dataset size.

In many existing retrieval methods, the similarity of two examples is measured by their Hamming distance. Here, similarity is rather measured by the number of leading bits they share. This is consistent with the training procedure, which produces codes with this property by demanding reconstructive ability under code truncation variation.

3.5.2 Empirical results

We empirically studied the properties of the resulting codes and data structures in a number of ways. First, we applied ordered retrieval to a toy problem where we trained a tiny 2-16-32-16-2 autoencoder on 2D synthetic pinwheel data (Figure 3-2). Here we can visualize the nesting of neighborhood families for different queries. Note that, as expected, the nested neighborhood boundaries are orthogonal to the direction of local variation of the data. This follows from the model's reconstruction loss function.

We then trained on 80MTI a binarized nested dropout autoencoder with layer widths 3072-2048-1024-512-1024-2048-3072 with ℓ_1 weight decay and invariance regularization (see Section 3.4). We chose $p_B(\cdot) \sim \text{Geom}(0.97)$ and the binarization quantile $\beta = 0.2$.

Empirical retrieval speeds for various models are shown in Figure 3-3. We performed retrieval by measuring Hamming distance in a linear scan over the database, and by means of semantic hashing for a number of radii. We also performed ordered retrieval for a number of terminal neighborhood cardinalities. Although semantic hashing is independent of the database size, for a radius greater than 2 it requires more time than a brute force linear scan even for very short codes. In addition, as the code length increases, it becomes very likely that many queries will not find any neighbors for any feasible radii. It can be seen that ordered retrieval carries a very small computational cost which is independent of the code length. Note that each multiplicative variation in the terminal neighborhood size R, from 2 to 32 to 512, leads to a constant shift downward on the logarithmic scale plot. This observation is consistent with our earlier analysis that the retrieval time increases logarithmically with N/R.

In Figure 3-4, we show retrieval results for varying terminal neighborhood sizes. As we decrease the terminal neighborhood size, the similarity of the retrieved data

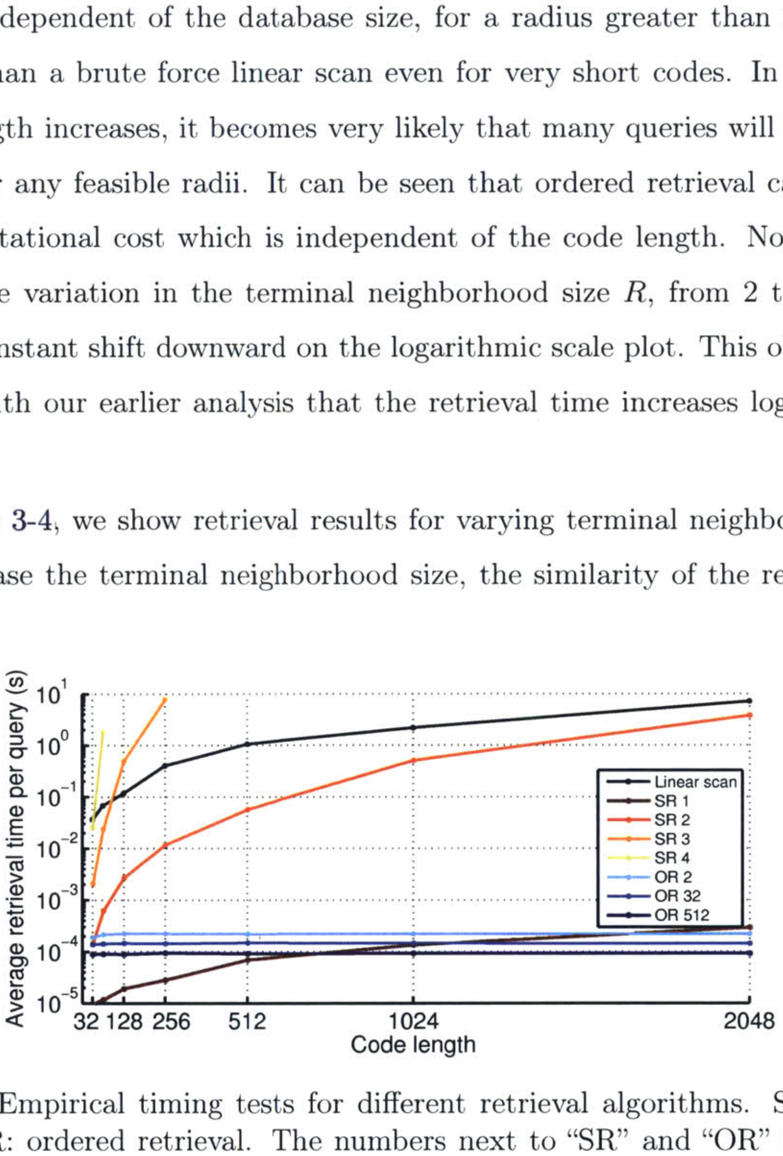


Figure 3-3: Empirical timing tests for different retrieval algorithms. SR: semantic retrieval, OR: ordered retrieval. The numbers next to "SR" and "OR" in the figure legend correspond to the semantic hashing radius and the terminal ordered retrieval neighborhood cardinality, respectively. As the code length increases, Hamming balls of very small radii become prohibitive to scan. Ordered retrieval carries a small fixed cost that is independent of code length.

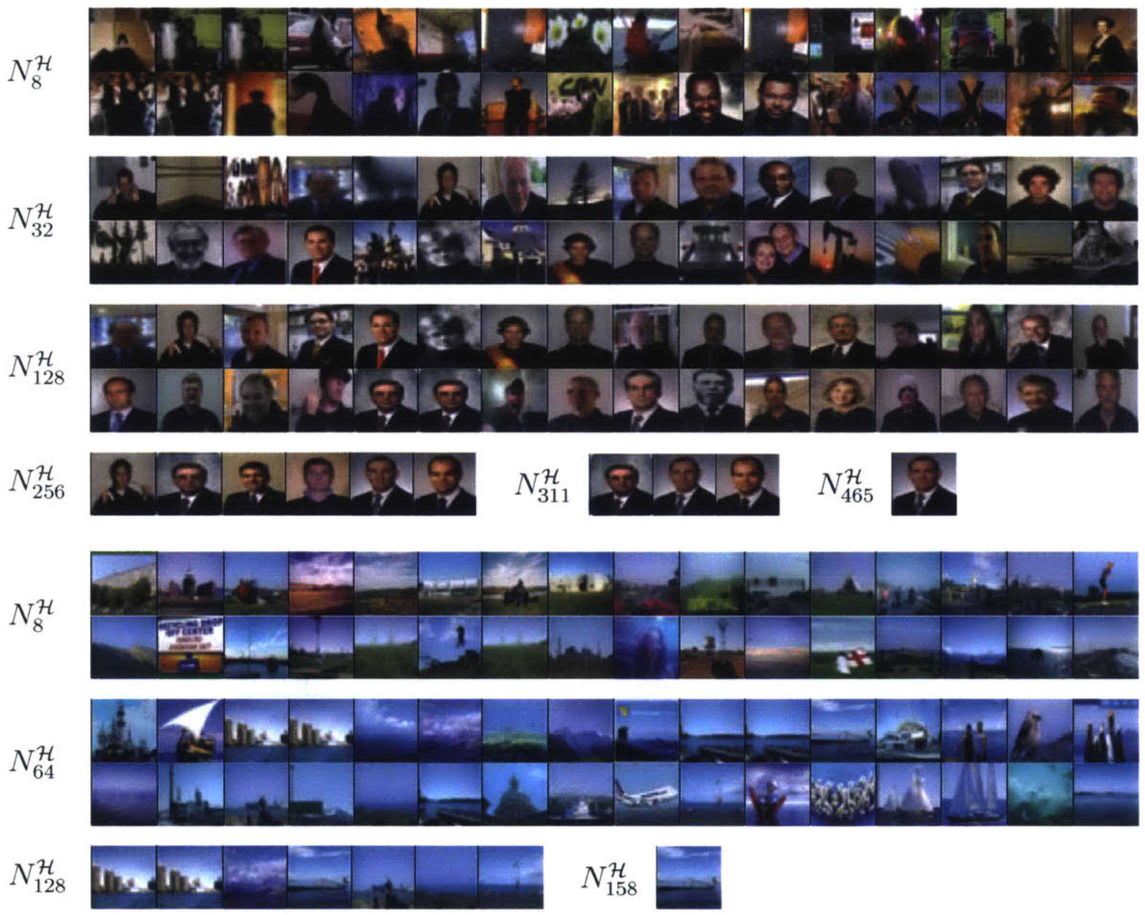


Figure 3-4: Retrieval results for different terminal neighborhood cardinalities. Note the increase in retrieval fineness as a function of neighborhood index. Examples presented in the order in which they appear in the dataset. When neighborhood sizes are greater than 32, only the first 32 images in the neighborhood are shown. The last neighborhood contains only the query itself.

to the query increases. As more bits are added to the representation in the process of retrieval, the resolution of the query increases, and thus it is better resolved from similar images.

3.6 Adaptive Compression

Another application of ordered representations is continuous-degradation lossy compression systems. By "continuous-degradation" we mean that the message can be decoded for any number, b, of bits received, and that the reconstruction error $\ell(\mathbf{y}, \hat{\mathbf{y}}_{\downarrow b})$

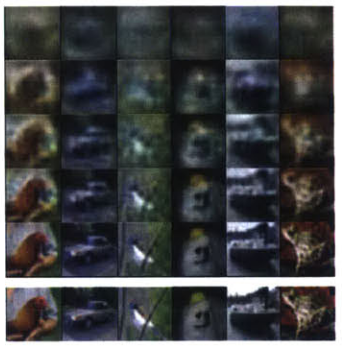
decreases monotonically with b. Such representations give rise to a continuous (up to a single bit) range of bitrate-quality combinations, where each additional bit corresponds to a small incremental increase in quality.

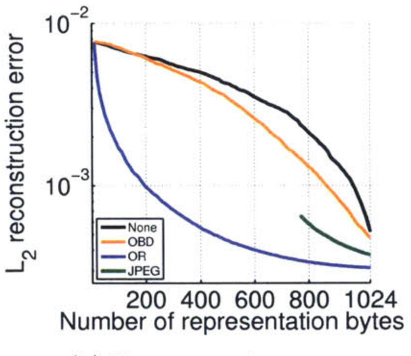
The continuous-degradation property is appealing in many situations. First, consider, a digital video signal that is broadcast to recipients with varying bandwidths. Assume further that the probability distribution over bandwidths for the population, $p_B(\cdot)$, is known or can be estimated, and that a recipient with bandwidth b receives only the first b bits of the transmission. We can then pose the following problem: what broadcast signal minimizes the expected distortion over the population? This is formulated as

$$(\boldsymbol{\Theta}^*, \boldsymbol{\Psi}^*) = \arg\min_{\boldsymbol{\Theta},\boldsymbol{\Psi}} \mathbb{E}_B \left[\ell\left(\mathbf{y}_n, \hat{\mathbf{y}}_{n\downarrow b}\right)\right] \ . \tag{3.15}$$

This is precisely the optimization problem solved by our model; Equation (**3.15**) is simply a rewriting of Equation (**3.3**). This connection gives rise to an interpretation of $p_B(\cdot)$, which we have set to the geometric distribution in our experiments. In particular, $p_B(\cdot)$ can be interpreted as the distribution over recipient bandwidths such that the system minimizes the expected reconstruction error.

This intuition in principle applies as well to online video streaming, in which the transmitted signal is destined for only a single recipient. Given that different recipients have different bandwidths, it is acceptable to lower the image quality in order to attain real-time video buffering. Currently, one may specify in advance a small number of fixed encodings for various bandwidths: for example, YouTube offers seven different definitions (240p, 360p, 480p, 720p, 1080p, 1440p, and 2160p), and automatically selects one of these to match the viewer's bitrate. Ordered representations offer the ability to fully utilize the recipient's bandwidth by truncating the signal to highest possible bitrate. Instead of compressing a handful of variants, one needs only to compute the ordered representation once in advance, and truncate it to the appropriate length at transmission time. If this desired bitrate changes over time, the quality could be correspondingly adjusted in a smooth fashion.


(a) Reconstructions



(b) Reconstruction rates

Figure 3-5: Online reconstruction with ordered representations. (a) Reconstructions for code lengths 16, 64, 128, 256, and 1024 using a nested dropout autoencoder. The original images are 24576 bits each. (b) Reconstruction rates as a function of code length for four different truncation techniques: ordered representation, Optimal Brain Damage, a standard autoencoder, and JPEG.

3.6.1 Empirical results

In Figure 3-5(a), we qualitatively evaluate continuous-degradation lossy compression with ordered representations. We trained a single-layer 3072-1024-3072 autoencoder with nested dropout on CIFAR-10, and produced reconstructions for different code lengths. Each column represents a different image and each row represents a different code length. As the code length increases (downwards in the figure), the reconstruction quality increases. The images second-to-bottom row look very similar to the original uncompressed images in the bottom row (24576 bits each).

Figure 3-5(b) shows ordered representation reconstruction rates as a function of code length for different approaches to the problem. In addition to the above, we also trained a standard autoencoder with the same architecture but without nested dropout. On this we applied 2 different truncation approaches. The first is a simple truncation on the un-ordered bits. The second is Optimal Brain Damage truncation (LeCun et al., 1990), which removes units in decreasing order of their influence on the reconstruction objective, measured in terms of the first and second order terms in its Taylor expansion. This is a clever way of ordering units, but is disjoint from the training procedure and is only applied retroactively. We also compare with JPEG compression. We use the libjpeg library and vary the JPEG quality parameter. Higher quality parameters result in larger file sizes and lower reconstruction error. Note that JPEG is not well-suited for the 32x32 pixel images we use in this study; its assumptions about the spectra of natural images are violated by such highly down-sampled images.

3.7 Discussion and Remaining Open Problems

We have presented a novel technique for learning representations in which the dimensions have a known ordering. This procedure is applicable to deep networks, and in the special case of shallow autoencoders is provably exactly equivalent to PCA. This enables learned representations of data that are adaptive in the sense that they can be truncated with the assurance that the shorter codes contain as much information as possible. Such codes are of interest in applications such as retrieval and compression.

The ordered representation retrieval approach can also be used for efficient supervised learning. Namely, it allows performing k-nearest-neighbors on very long codes in logarithmic time in their cardinality. This idea can be combined with various existing approaches to metric learning of kNN and binarized representations (Norouzi et al., 2012; Salakhutdinov & Hinton, 2007; Weinberger & Saul, 2009). The purely unsupervised approaches we have described here have not been empirically competitive with state of the art supervised methods from deep learning. We are optimistic that nested dropout can be meaningfully combined with supervised learning, but leave this for future work.

In addition, ordered representations provide a practical way to train models with an infinite number of latent dimensions, in the spirit of Bayesian nonparametric methods. For example, the distribution $p_B(\cdot)$ can be chosen to have infinite support, while having finite mean and variance.

Chapter 4

Metric Learning with Adaptive Density Discrimination

4.1 Introduction

The problem of classification is a mainstay task in machine learning, as it provides us with a coherent metric to gauge progress and juxtapose new ideas against existing approaches. To tackle various other tasks beyond categorization, we often require alternative representations of our inputs which provide succinct summaries of relevant characteristics. Here, classification algorithms often serve as convenient feature extractors: a very popular approach involves training a network for classification on a large dataset, and retaining the outputs of the last layer as inputs transferred to other tasks (Donahue et al.; Sharif Razavian et al., 2014; Qian et al., 2015; Snoek et al., 2015).

However, this paradigm exhibits an intrinsic discrepancy: we have no guarantee that our extracted features are suitable for any task but the particular classification problem from which they were derived. On the contrary: in our classification procedure, we propagate high-dimensional inputs through a complex pipeline, and map each to a single, scalar prediction. That is, we explicitly demand our algorithm to, ultimately, dispose of all information but class label. In the process, we destroy intra- and inter-class variation that would in fact be desirable to maintain in our features.

In principle, we have no reason to compromise: we should be able to construct a representation which is amenable to classification, while still maintaining more fine-grained information. This philosophy motivates the class of distance metric learning (DML) approaches, which learn a transformation to a representation space where distance is in correspondence with a notion of similarity. Metric learning offers a number of benefits: for example, it enables zero-shot learning (Mensink et al., 2013; Chopra et al., 2005), visualization of high-dimensional data (van der Maaten & Hinton, 2008), learning invariant maps (Hadsell et al., 2006), and graceful scaling to instances with millions of classes (Schroff et al., 2015). In spite of this, it has been difficult for DML-based approaches to compete with modern classification algorithms in performance and even in feature extraction.

Admittedly, however, these are two sides of the same coin: a more salient representation should, in theory, enable improved classification performance and features for task transfer. In this work, we strive to reconcile this gap. We introduce *Magnet Loss*, a novel approach explicitly designed to address subtle yet important issues which have hindered the quality of learnt representations and the training efficiency of a class of DML approaches. In essence, instead of penalizing individual examples or triplets, it maintains an explicit model of the distributions of the different classes in representation space. It then employs this knowledge to adaptively assess similarity, and achieve discrimination by reducing local distribution overlap. It utilizes clustering techniques to simultaneously tackle a number of components in model design, from capturing the distributions of the different classes to hard negative mining. For a particular set of assumptions in its configuration, it reduces to the familiar triplet loss (Weinberger & Saul, 2009).

We demonstrate the effectiveness of this idea on several tasks. Using a soft k-nearest-cluster metric for evaluation, this approach achieves state-of-the-art classification results on a number of fine-grained visual recognition datasets, surpassing the standard softmax classifier and outperforming triplet loss by a relative margin of 30-40%. In terms of computational performance, it alleviates several training inefficiencies in traditional triplet-based approaches, reaching the same error in 5-30 times

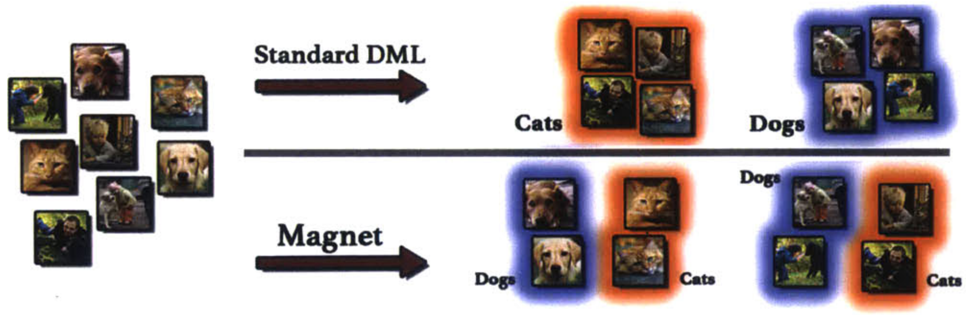


Figure 4-1: Distance metric learning approaches sculpt a representation space where distance is in correspondence with a notion of similarity. Traditionally, similarity is specified *a-priori* and often strictly semantically. In contrast, Magnet Loss adaptively sculpts its representation space by autonomously identifying and respecting intra-class variation and inter-class similarity.

fewer iterations. Beyond classification, we further validate the saliency of the learnt representations via their attribute concentration and hierarchy recovery properties, achieving 10-25% relative gains on the softmax classifier and 25-50% on triplet loss in these tasks.

4.2 Motivation: Challenges in Metric Learning

We start by providing an overview of challenges which we believe have been impeding the success of existing distance metric learning approaches. These will motivate our work to follow.

ISSUE #1: PREDEFINED TARGET NEIGHBOURHOOD STRUCTURE All metric learning approaches must define a relationship between similarity and distance, which prescribes neighbourhood structure. The corresponding training algorithm, then, learns a transformation to a representation space where this property is obeyed. In existing approaches, similarity has been canonically defined *a-priori* by integrating available supervised knowledge. The most common is semantic, informed by class labels. Finer assignment of neighbourhood structure is enabled with access to additional prior information, such as similarity ranking (Wang et al., 2014) and hierarchical class taxonomy (Verma et al., 2012).

In practice, however, the only available supervision is often in the form of class labels. In this case, a ubiquitous solution is to enforce semantic similarity: examples of each class are demanded to be tightly clustered together, far from examples of other classes (for example, **Schroff et al. (2015)**; **Norouzi et al. (2012)**; **Globerson & Roweis (2006)**; **Chopra et al. (2005)**). However, this collapses intra-class variation and does not embrace shared structure between different classes. Hence, this imposes too strong of a requirement, as each class is assumed to be captured by *a single mode*.

This issue is well-known, and has motivated the notion of *local* similarity: each example is designated only a small number of target neighbours of the same class (**Weinberger & Saul, 2009**; **Qian et al., 2015**; **Hadsell et al., 2006**). In existing work, these target neighbours are determined *prior to training*: they are retrieved based on distances in the *original input space*, and after which are never updated again. Ironically, this is in contradiction with our fundamental assumption which motivated us to pursue a DML approach in the first place. Namely, we want to learn a metric because we cannot trust distances in our original input space — but on the other hand define target similarity using this exact metric that cannot be trusted! Thus, although this approach has the good intentions of encoding similarity into our representation, it harms intra-class variation and inter-class similarity by enforcing unreasonable proximity relationships. Apart from its information preservation ramifications, achieving predefined separation requires significant effort, which results in inefficiencies during training time.

Instead, what we ought to do is rather define similarity as function of distances of our *representations* — which lie in precisely the space sculpted for metric saliency. Since representations are adjusted continuously during training, it then follows that similarity must be defined adaptively. To that end, we must alternate between updating our representations, and refreshing our model which designates similarity as function of these. Visualizations of representations of different DML approaches can be found in a toy example in Figure 4-2.

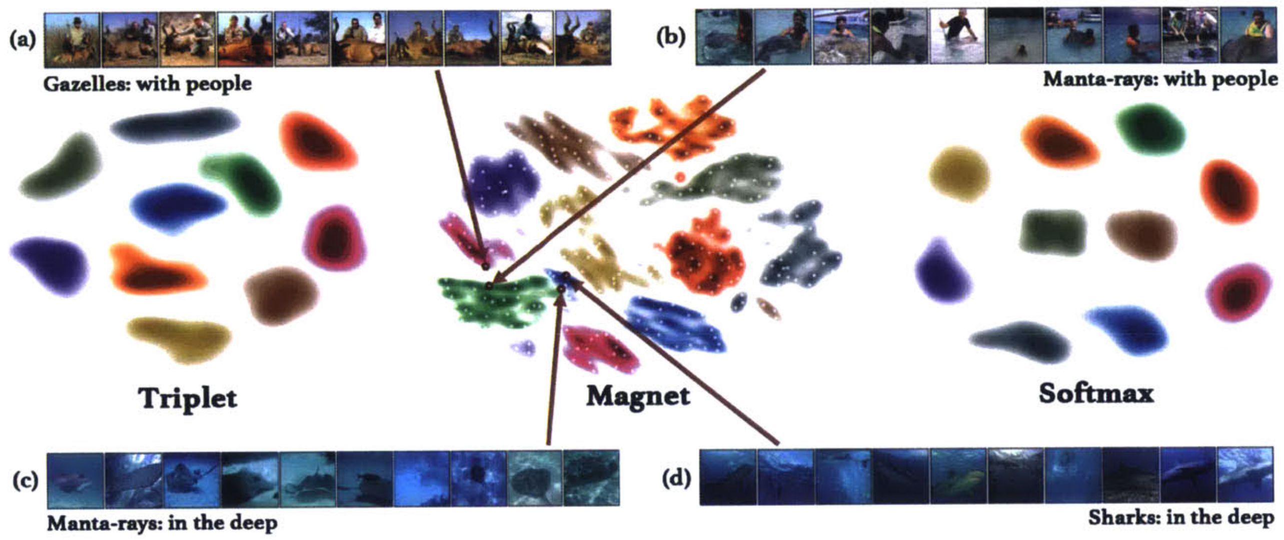
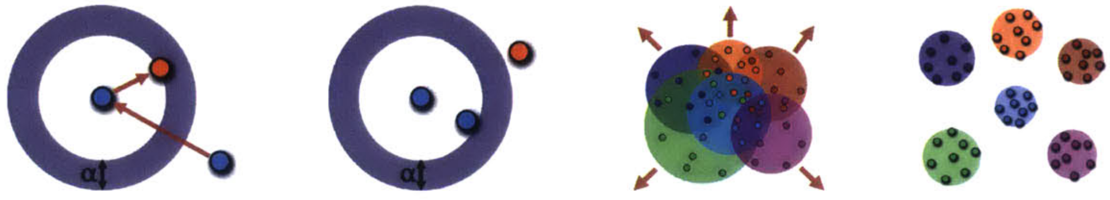


Figure 4-2: 2D visualizations of representations attained by training triplet loss, Magnet Loss and a softmax classifier on 10 classes of ImageNet. The **different colours** correspond to different classes, and the values to density estimates computed from an application of t-SNE (van der Maaten & Hinton, 2008) on the original 1024-dimensional representations. The **white dots** in the Magnet t-SNE correspond to $K = 32$ clusters used by Magnet to capture each class. The **red arrows** retrieve the examples closest to particular clusters (which were learnt autonomously). **1.** It can be seen that triplet loss and softmax result in unimodal separation, due to enforcement of semantic similarity. For Magnet Loss, the distributions of the different classes may arbitrarily split, adaptively embracing intra-class variation and inter-class similarity. **2.** Green corresponds to manta-rays, blue to sharks, and magenta to gazelles. Magnet Loss captures intra-class variation between (c) and (b) as manta-rays in the deep, and manta-rays with people. It also respects inter-class similarity, allowing shared structure between (c) and (d) as fish in the deep, and between (a) and (b) as animals with people. See Figures 4-6 and 4-7 for image maps of other t-SNE projections.

ISSUE #2: OBJECTIVE FORMULATION Two very popular classes of DML approaches have stemmed from Triplet Loss (Weinberger & Saul, 2009) and Contrastive Loss (Hadsell et al., 2006). The outlined issues apply to both, but for simplicity of exposition we use triplet loss as an example. During its training, triplets consisting of a seed example, a "positive" example similar to the seed and a "negative" dissimilar example are sampled. Let us denote their representations as $\mathbf{r}_m, \mathbf{r}_m^+$ and $\mathbf{r}_m^-$ for $m = 1, \ldots, M$. Triplet loss then demands that the difference of distances of the representation of the seed to the negative and to the positive be larger than some pre-assigned margin



(a) Triplet: before. (b) Triplet: after. (c) Magnet: before. (d) Magnet: after.

Figure 4-3: The intuition behind triplet loss and Magnet Loss. Triplet loss only considers a single triplet at a time, resulting in reduced performance and training inefficiencies. In contrast, in Magnet Loss, at each iteration an entire local neighbourhood of nearest clusters is retrieved, and their overlaps are penalized. Insight into representation distribution permits adaptive similarity characterization, local discrimination and a globally consistent optimization procedure.

constant $\alpha \in \mathbb{R}$:

$$\mathscr{L}_{\text{triplet}}(\mathbf{\Theta}) = \frac{1}{M} \sum_{m=1}^{M} \left\{ \left\| \mathbf{r}_m - \mathbf{r}_m^- \right\|_2^2 - \left\| \mathbf{r}_m - \mathbf{r}_m^+ \right\|_2^2 + \alpha \right\}_+ , \tag{4.1}$$

where $\{\cdot\}_+$ is the hinge function and $\mathbf{\Theta}$ the parameters of the map to representation space. The representations are often normalized to achieve scale invariance, and negative examples are mined in order to find margin violators (for example, Schroff et al. (2015); Norouzi et al. (2012)).

Objectives formulated in this spirit exhibit a short-sightedness. Namely, penalizing individual pairs or triplets of examples does not employ sufficient contextual insight of neighbourhood structure, and as such different triplet terms are not necessarily consistent. This hinders both the convergence rate as well as performance of these approaches. Moreover, the cubic growth of the number of triplets renders operation on these computationally inefficient.

In contrast to this, it is desirable to instead inform the algorithm of the *distributions* of the different classes in representation space and their overlaps, and rather manipulate these in a way that is globally consistent. We elaborate on this in the section below.

4.3 Magnet Loss for Distance Metric Learning

We proceed to design a model to mitigate the identified difficulties. Let us for a moment neglect practical considerations, and envision our ideal DML approach. To start, as concluded at the start of Section 4.2, we are interested to characterize similarity adaptively as function of current representation structure. We would then utilize this knowledge to pursue local separation as opposed to global: we seek to separate between distributions of different classes in representation space, but do not mind if they are interleaved. As such, let us assume that we have knowledge of the representation distribution of each class at any time during training. Our DML algorithm, then, would discover regions of local overlap between different classes, and penalize these to achieve discrimination.

Such an approach would liberate us from the unimodality assumption and unreasonable prior target neighbourhood assignments — resulting in a more expressive representation which maintains significantly more information. Moreover, employing a loss informed of distributions rather than individual examples would allow for a more coherent training procedure, where the distance metric is adjusted in a way that is globally consistent.

To that end, a natural approach would be to employ clustering techniques to capture these distributions in representation space. Namely, for each class, we will maintain an index of clusters, which we will update continuously throughout training. Our objective, then, would jointly manipulate entire clusters — as opposed to individual examples — in the pursuit of local discrimination. This intuition of cluster attraction and repulsion motivates us to name it *Magnet Loss.* A caricature illustrating the intuition behind this approach can be found in Figure 4-3.

In addition to its advantages from a modeling perspective, a clustering-based approach also facilitates computation by enabling efficient hard negative mining. That is, we may perform approximate nearest neighbour retrieval in a two-step process, where we first retrieve nearest clusters, after which we retrieve examples from these clusters.

Finally, as discussed, throughout training we are interested in a more complete characterization of neighbourhood structure. At each iteration, we sample *entire local neighbourhoods* rather than collections of independent examples (or triplets) as per usual, which significantly improves training efficiency. We elaborate on this in Section 4.3.2.

4.3.1 Model formulation

We proceed to quantify the modeling objectives outlined above. Let us assume we have a training set consisting of N input-label pairs $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^{N}$ belonging to C classes. We consider a parametrized map $\mathbf{f}(\cdot; \mathbf{\Theta})$ which hashes our inputs to *representation space*, and denote their representations as $\mathbf{r}_n = \mathbf{f}(\mathbf{x}_n; \mathbf{\Theta}), n = 1, \ldots, N$. In this work, we select this transformation as GoogLeNet (Szegedy et al., 2015; Ioffe & Szegedy, 2015a), which has been demonstrated to be a powerful CNN architecture; in Section 4.4 we elaborate on this choice.

We assume that, for each class c, we have K cluster assignments $\mathcal{I}_1^c, \ldots, \mathcal{I}_K^c$ obtained via an application of the K-means algorithm. Note that K may vary across classes, but for simplicity of exposition we fix it as uniform. In Section 4.3.2, we discuss how to maintain this index. To that end, we assume that these assignments have been chosen to minimize intra-cluster distances. Namely, for each class c, we have

$$\mathcal{I}_1^c, \ldots, \mathcal{I}_K^c = \arg \min_{I_1^c, \ldots, I_K^c} \sum_{k=1}^{K} \sum_{\mathbf{r} \in I_k^c} \|\mathbf{r} - \boldsymbol{\mu}_k^c\|_2^2 \,, \tag{4.2}$$

$$\boldsymbol{\mu}_k^c = \frac{1}{|I_k^c|} \sum_{\mathbf{r} \in I_k^c} \mathbf{r} \,. \tag{4.3}$$

We further define $C(\mathbf{r})$ as the class of representation $\mathbf{r}$, and $\boldsymbol{\mu}(\mathbf{r})$ as its assigned cluster center.

We proceed to define our objective as follows:

$$\mathscr{L}(\Theta) = \frac{1}{N}\sum_{n=1}^{N}\left\{-\log\frac{e^{-\frac{1}{2\sigma^2}\|\mathbf{r}_n-\boldsymbol{\mu}(\mathbf{r}_n)\|_2^2-\alpha}}{\sum_{c\neq C(\mathbf{r}_n)}\sum_{k=1}^{K} e^{-\frac{1}{2\sigma^2}\|\mathbf{r}_n-\boldsymbol{\mu}_k^c\|_2^2}}\right\}_+ \tag{4.4}$$

where $\{\cdot\}_+$ is the hinge function, $\alpha \in \mathbb{R}$ is a scalar, and $\sigma^2 = \frac{1}{N-1}\sum_{\mathbf{r}\in\mathcal{D}}\|\mathbf{r}-\boldsymbol{\mu}(\mathbf{r})\|_2^2$ is the variance of all examples away from their respective centers. We note that cluster centers sufficiently far from a particular example vanish from its term in the objective. This allows accurately approximating each term with a small number of nearest clusters.

A feature of this objective not usually available in standard distance metric learning approach is variance standardization. This renders the objective invariant to the characteristic lengthscale of the problem, and allows the model to gauge its confidence of prediction by comparison of intra- and inter-cluster distances. With this in mind, α is then the desired cluster separation gap, measured in units of variance. In our formulation, we may thus interpret α as a modulator of the probability assigned to an example of a particular class under the distribution of another.

We remark that during model design, an alternative objective we considered is the cluster-based analogue of NCA (see Section 4.3.4): this objective seems to be a natural approach with a clear probabilistic interpretation. However, we found empirically that this objective does not generalize as well, since it only vanishes in the limit of extreme discrimination margins.

4.3.2 Training procedure

COMPONENT #1: NEIGHBOURHOOD SAMPLING At each iteration, we sample entire local neighbourhoods rather than a collection of independent examples. Namely, we construct our minibatch in the following way:

1. Sample a seed cluster $I_1 \sim p_{\mathcal{I}}(\cdot)$

2. Retrieve $M-1$ nearest impostor clusters $I_2, \ldots, I_M$ of I_1

3. For each cluster $I_m, m = 1, \dots, M$, sample D examples $\mathbf{x}_1^m, \dots \mathbf{x}_D^m \sim p_{I_m}(\cdot)$

The choices of $p_{\mathcal{I}}(\cdot)$ and $p_{I_m}(\cdot), m = 1, \dots, M$ allow us to adapt to the current distributions of examples in representation space. Namely, in our training, these allow us to specifically target and reprimand contested neighbourhoods with large cluster overlap. During training, we cache the losses of individual examples, from which we compute the mean loss $\mathscr{L}_I$ of each cluster I. We then choose $p_{\mathcal{I}}(I) \propto \mathscr{L}_I$, and $P_{I_m}(\cdot)$ as a uniform distribution. We remark that these choices work well in practice, but have been made arbitrarily and perhaps can be improved.

Given our samples, we may proceed to construct a stochastic approximation of our objective:

$$\hat{\mathscr{L}}(\Theta) = \frac{1}{MD} \sum_{m=1}^{M} \sum_{d=1}^{D} \left\{ -\log \frac{e^{-\frac{1}{2\hat{\sigma}^2} \left\| \mathbf{r}_d^m - \hat{\boldsymbol{\mu}}_m \right\|_2^2 - \alpha}}{\sum_{\hat{\boldsymbol{\mu}} : C(\hat{\boldsymbol{\mu}}) \neq C(\mathbf{r}_d^m)} e^{-\frac{1}{2\hat{\sigma}^2} \left\| \mathbf{r}_d^m - \hat{\boldsymbol{\mu}} \right\|_2^2}} \right\}_+ \tag{4.5}$$

where we approximate the cluster means as $\hat{\boldsymbol{\mu}}_m = \frac{1}{D} \sum_{d=1}^{D} \mathbf{r}_d^m$ and variances as $\hat{\sigma} = \frac{1}{MD-1} \sum_{m=1}^{M} \sum_{d=1}^{D} \|\mathbf{r}_d^m - \hat{\boldsymbol{\mu}}_m\|_2^2$. During training, we backpropagate through this objective, and the full CNN which gave rise to the representations.

COMPONENT #2: CLUSTER INDEX As mentioned above, we maintain for each class a K-means index which captures its distribution in representation space during training. We initialize each index with K-means++ (Arthur & Vassilvitskii, 2007), and refresh it periodically. To attain the representations to be indexed, we pause training and compute the forward passes of all inputs in the training set.

It may seem that freezing the training is unnecessarily computationally expensive. Note that we also explored the alternative strategy of caching the representations of each minibatch on-the-fly during training. However, we found that it is critical to maintain the true neighbourhood structure where the representations are all computed in the same stage of learning. We empirically observed that since the representation space is changing continuously during training, indexing examples whose representations were computed in different times resulted in incorrect inference of neighbourhood structure, which in turn irreparably damaged nearest impostor assessment.

Improvement of training efficiency The proposed approach offers a number of benefits which compound to considerably enhance training efficiency, as can be seen empirically in Section 4.4.1. First, one of the main criticisms of triplet-based approaches is the cubic growth of the number of triplets. Manipulating entire clusters of examples, on the other hand, significantly improves this complexity, as this requires far fewer pairwise distance evaluations. Second, operating on entire cluster neighbourhoods also permits information recycling: we may jointly separate all clusters from one another at once, whereas an approach based on independent sampling would require far more repetitions of the same examples. Finally, penalizing clusters of points away from one another leads to a more coherent adjustment of each point, whereas different triplet terms may not necessarily be consistent with one another.

4.3.3 Evaluation procedure

The evaluation procedure is consistent with the objective formulation: we assign the label of each example $\mathbf{x}_n$ as function of its representation's softmax similarities to its L closest clusters, say $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_L$. More precisely, we choose label c_n^* as

$$c_n^* = \arg\max_{c=1,\ldots,C} \frac{\sum_{\boldsymbol{\mu}_l : C(\boldsymbol{\mu}_l)=c} e^{-\frac{1}{2\sigma^2}\|\mathbf{r}_n - \boldsymbol{\mu}_l\|_2^2}}{\sum_{l=1}^{L} e^{-\frac{1}{2\sigma^2}\|\mathbf{r}_n - \boldsymbol{\mu}_l\|_2^2}} , \tag{4.6}$$

where σ is a running average of stochastic estimates $\hat{\sigma}$ computed during training.

This can be thought of as "k-nearest-cluster" (kNC), a variant of a soft kNN classifier. This has the added benefit of reducing the complexity of nearest neighbour evaluation from being a function of the number of examples to the number of clusters. Here, the lengthscale σ autonomously characterizes local neighbourhood radius, and as such implies how to sensibly choose L. In general, we found that performance improves monotonically with L, as the soft classification is able to make use of additional neighbourhood information. At some point, however, retrieving additional nearest neighbours is clearly of no further utility, since these are much farther away than the lengthscale defined by σ. In practice we use $L = 128$ for all experiments in this work.

4.3.4 Relation to existing models

TRIPLET LOSS Our objective proposed in Equation 4.4 has the nice property that it reduces to the familiar triplet loss under a particular set of assumptions. Specifically, let us assume that we approximate each neighbourhood with a *single* impostor cluster, i.e, $M = 2$. Let us further assume that we approximate the seed cluster with merely $D = 2$ samples, and the impostor cluster with one. We further simplify by ignoring the variance normalization. Our objective then exactly reduces to triplet loss for a pair of triplets "symmetrized" for the two positive examples:

$$\hat{\mathscr{L}}(\Theta) = \sum_{d=1}^{2} \left\{ \left\|\mathbf{r}_d^1 - \mathbf{r}_{2-d}^1\right\|_2^2 - \left\|\mathbf{r}_d^1 - \mathbf{r}_1^2\right\|_2^2 + \alpha \right\}_+ . \tag{4.7}$$

NEIGHBOURHOOD COMPONENTS ANALYSIS Neighbourhood Components Analysis (NCA) and its extensions (Goldberger et al., 2004; Salakhutdinov & Hinton, 2007; Min et al., 2010) have been designed in a similar spirit to Magnet Loss. The NCA objective is given by

$$\mathscr{L}_{\text{NCA}}(\Theta) = \frac{1}{N} \sum_{n=1}^{N} -\log \frac{\sum_{n' : C(\mathbf{r}_{n'}) = C(\mathbf{r}_n)} e^{-\|\mathbf{r}_n - \mathbf{r}'_n\|_2^2}}{\sum_{n'=1}^{N} e^{-\|\mathbf{r}_n - \mathbf{r}'_n\|_2^2}} . \tag{4.8}$$

However, this formulation does not address a number of concerns both in modeling and implementation. As an example, it does not touch on minibatch sampling in large datasets. Even if we maintain a nearest neighbour index, if we naïvely retrieve the nearest neighbours for each example, they are all going to be of different classes with high probability.

NEAREST CLASS MEAN Our approach shares many ideas with Nearest Class Mean (Mensink et al., 2013), cleverly designed for scalable DML. In this approach, the mean vectors $\boldsymbol{\mu}_c = \frac{1}{|C|} \sum_{c(\mathbf{x})=c} \mathbf{x}$, $c = 1, \ldots, C$ of the examples in their raw input form are computed and fixed for each class. A linear transformation $\mathbf{W}$ is then learned to maximize the softmax distance of each example to the cluster center of its class:

$$\mathscr{L}_{\text{NCM}}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N} - \log \frac{e^{-\|\mathbf{W}\mathbf{x}_n - \mathbf{W}\boldsymbol{\mu}_{c(\mathbf{x}_n)}\|_2^2}}{\sum_{c=1}^{C} e^{-\|\mathbf{W}\mathbf{x}_n - \mathbf{W}\boldsymbol{\mu}_c\|_2^2}} \,. \tag{4.9}$$

The authors further generalize this to Nearest Class Multiple Centroids (NCMC), where for each class, K centroids are computed with K-means. Note that these are computed on the raw inputs, and are fixed prior to training; also, in this case, the underlying map to representation space is a linear transformation rather than a full CNN.

4.4 Experiments

We run all experiments on a cluster of Tesla K40M GPU's. All parametrized maps $\mathbf{f}(\cdot;\boldsymbol{\Theta})$ to representation space are chosen as GoogLeNet with batch normalization (**Ioffe & Szegedy, 2015a**). We add an additional fully-connected layer to map to a representation space of dimension 1024.

We find that it is useful to warm-start any DML optimization with weights of a partly-trained a standard softmax classifier. It is important to not use weights of a net trained to completion, as this would result in information dissipation and as such defeat the purpose of pursuing DML in the first place. Hence, we initialize all DML models with weights of a net trained on ImageNet (**Russakovsky et al., 2015**) for 3 epochs only. We augment all experiments with random input rescaling of up to 30%, followed by jittering back to the original input size of 224×224. At test-time we evaluate an input by averaging the outputs of 16 random samples drawn from this augmentation distribution.

4.4.1 Fine-grained classification

We validate the classification efficacy of the learnt representations on a number of popular fine-grained visual categorization tasks, including Stanford Dogs (**Khosla**

Approach	Error
Angelova & Long	51.7%
Gavves et al.	49.9%
Xie et al.	43.0%
Gavves et al.	43.0%
Qian et al.	30.9%
Softmax	26.6%
Triplet	35.8%
Magnet	**24.9%**

(a) Stanford Dogs.

Approach	Error
Angelova & Zhu	23.3%
Angelova & Long	19.6%
Murray & Perronnin	15.4%
Sharif Razavian et al.	13.2%
Qian et al.	11.6%
Softmax	11.2%
Triplet	17.0%
Magnet	**8.6%**

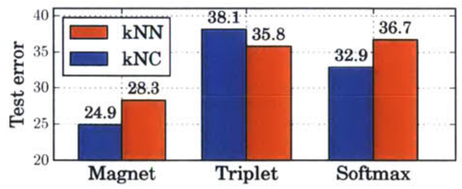
(b) Oxford 102 Flowers.

Approach	Error
Angelova & Zhu	49.2%
Parkhi et al.	46.0%
Angelova & Long	44.6%
Murray & Perronnin	43.2%
Qian et al.	19.6%
Softmax	11.3%
Triplet	13.5%
Magnet	**10.6%**

(c) Oxford-IIIT Pet.

Approach	Error
Softmax	**14.1%**
Triplet	26.8%
Magnet	15.9%

(d) ImageNet Attributes.



(e) Different metrics on Stanford Dogs.

Approach	Error@1	Error@5
Softmax	30.9%	15.0%
Triplet	44.6%	23.4%
Magnet	**28.6%**	**7.8%**

(f) Hierarchy recovery on ImageNet Attributes.

Figure 4-4: **(a)-(d)** Comparison of test set errors of various state-of-the-art approaches on different fine-grained visual categorization datasets. The bottom three results for each table were all attained by applying different objectives on exactly the same architecture. **(e)** Evaluation of test errors on the Stanford Dogs dataset under different metrics. **(f)** We explore whether each algorithm is able to recover a latent class hierarchy, provided only coarse superclasses. We collapse random pairs of classes of ImageNet Attributes onto the same label. We then train on the corrupted labels, and report test errors on the original classes.

et al., 2011), Oxford-IIIT Pet (Parkhi et al., 2012) and Oxford 102 Flowers (Nilsback & Zisserman, 2008) datasets. We also include results on ImageNet attributes, a dataset described in Section 4.4.2.

We seek to compare optimal performances of the different *model spaces*, and so perform hyperparameter search on validation error generated by 3 classes of objectives: a standard softmax classifier, triplet loss, and Magnet Loss. The hyperparameter search setup, including optimal configurations for each experiment, is specified in full detail in Appendix B.1. In general, for Magnet Loss we observed empirically that it is beneficial to increase the number of clusters per minibatch to around $M = 12$ in

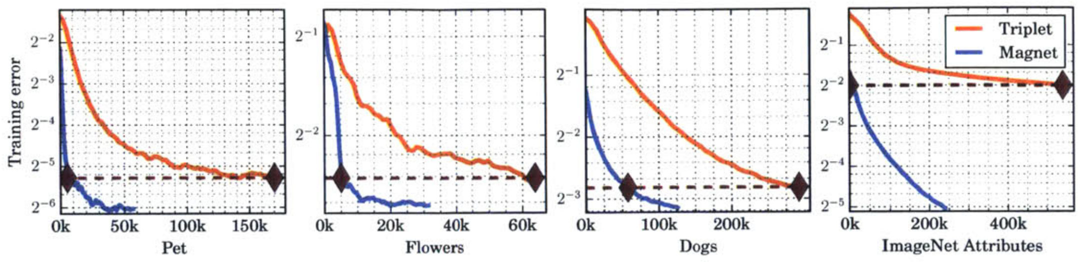


Figure 4-5: Training curves for various experiments. For both triplet and Magnet Loss objectives, the experiment with optimal hyperparameter configuration for each model space is presented. The red diamonds indicate the point in time in which the triplet asymptotic error rate is achieved. It can be observed that Magnet Loss reaches the same error in 5-30 times fewer iterations.

the cost of reducing the number of retrieved examples per cluster to $D = 4$. The optimal gap has in general been $\alpha \approx 1$, and the value of K varied as function of dataset cardinality.

The classification results can be found in Table 4-4. We use soft kNN to evaluate triplet loss error and kNC (see Section 4.3.3) for Magnet Loss. However, for completeness of comparison, in Figure 4-4(e) we present evaluations of all learnt representations under both kNN and kNC.

It can be observed that Magnet Loss outperforms the traditional triplet loss by a considerable margin. It is also able to surpass the standard softmax classifier in most cases: while the margin is not significant, note that the true win here is in terms of learning representations much more suitable for task transfer, as validated in the following subsections.

In Figure 4-5, it can be seen that Magnet Loss reaches the triplet loss asymptotic error rate 5-30 times faster. The prohibitively slow convergence of triplet loss has been well-known in the community. Magnet Loss achieves this speedup as it mitigates some of the training-time inefficiencies featured by triplet loss presented throughout Section 4.2 and the end of Section 4.3.2. For fairness of comparison, we remark that softmax converges faster than Magnet; however, this comes at the cost of a less informative representation.

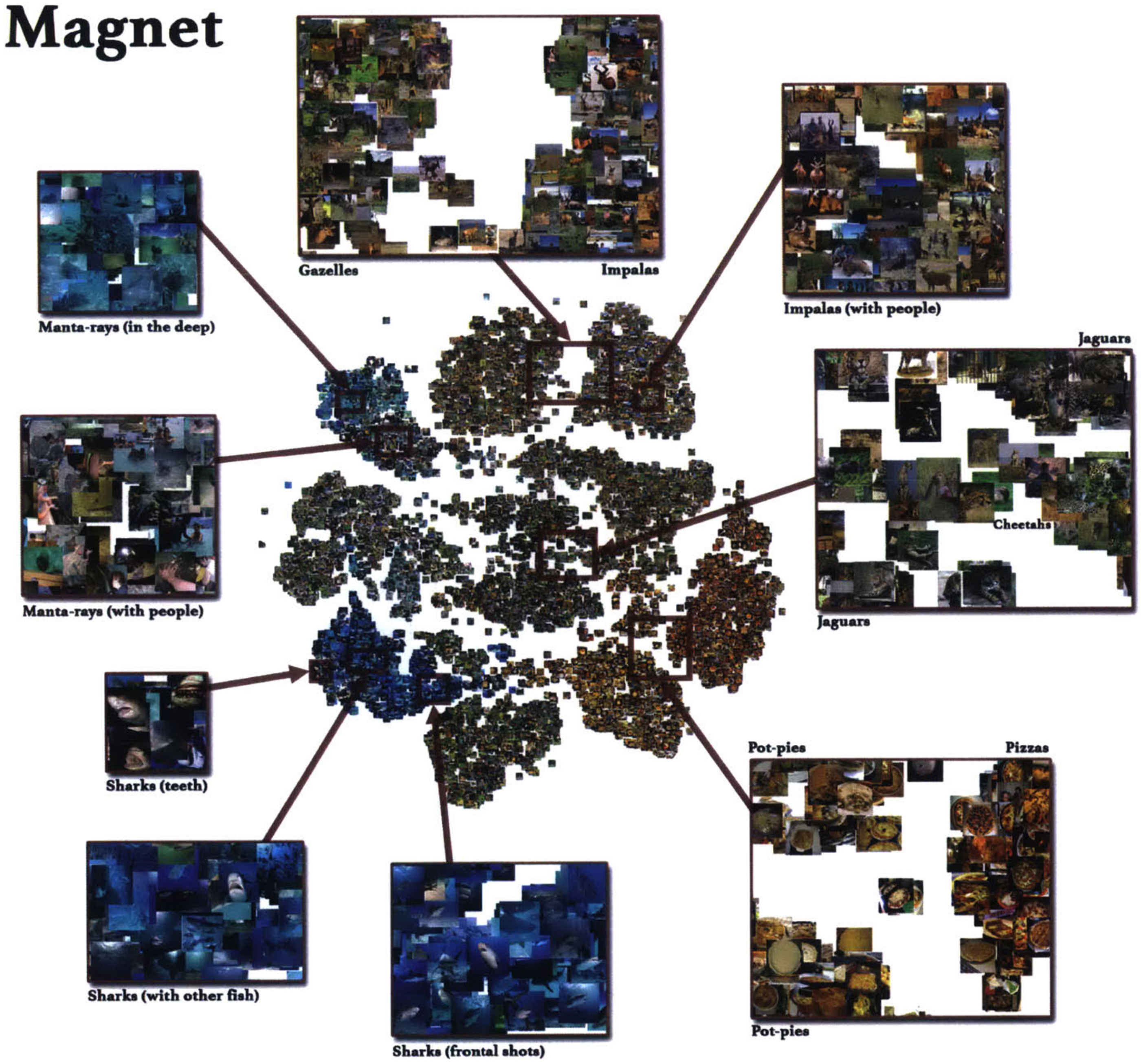


Figure 4-6: Visualization of t-SNE map for a typical Magnet representation. We highlight interesting observations of the distributions of the learnt representations splitting to repsect intra-class variance and inter-class similarity.

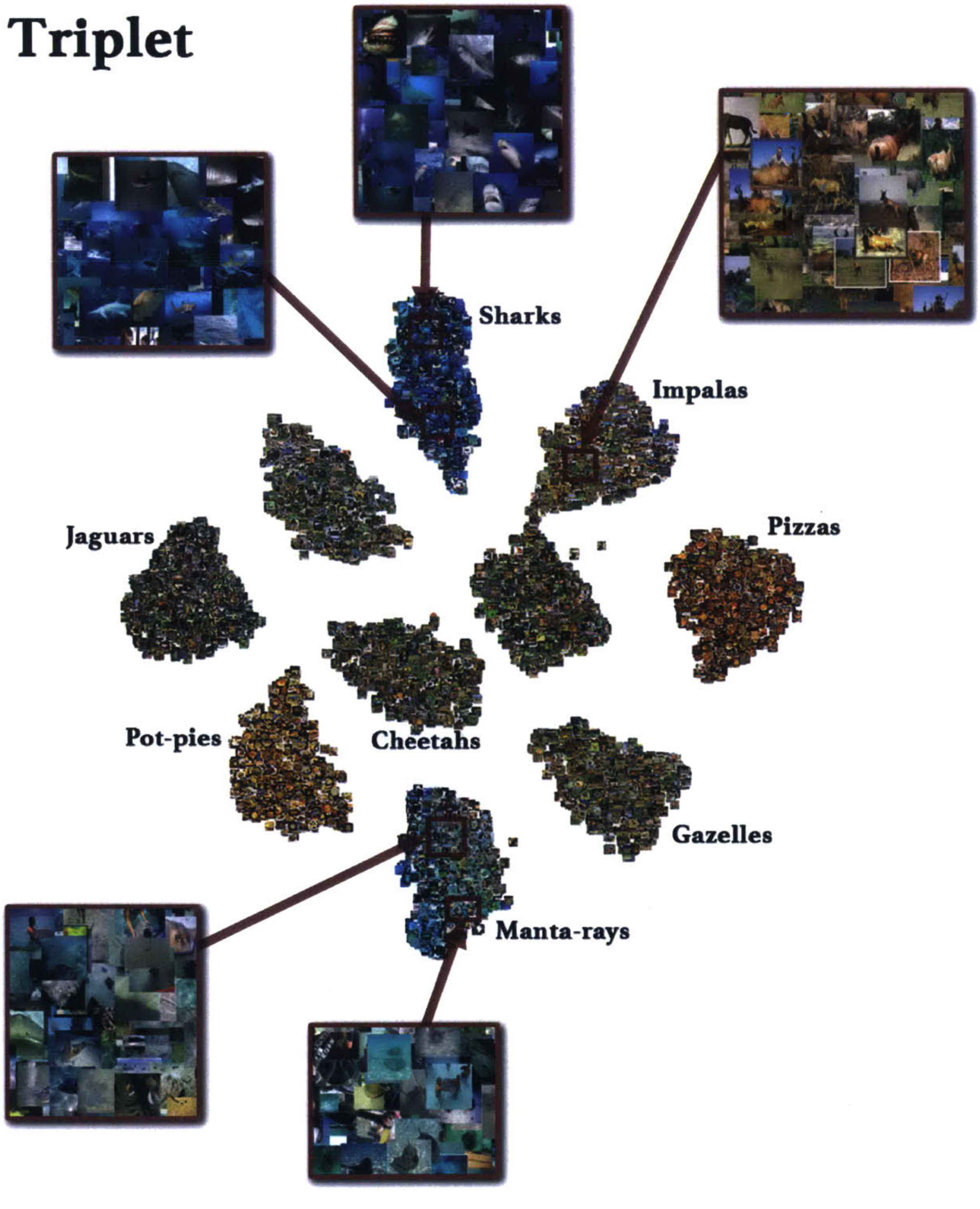


Figure 4-7: Visualization of t-SNE map for a typical triplet representation with enforcement of semantic similarity. Classes with similar examples are far from one another, and no obvious local similarity can be found within individual classes.

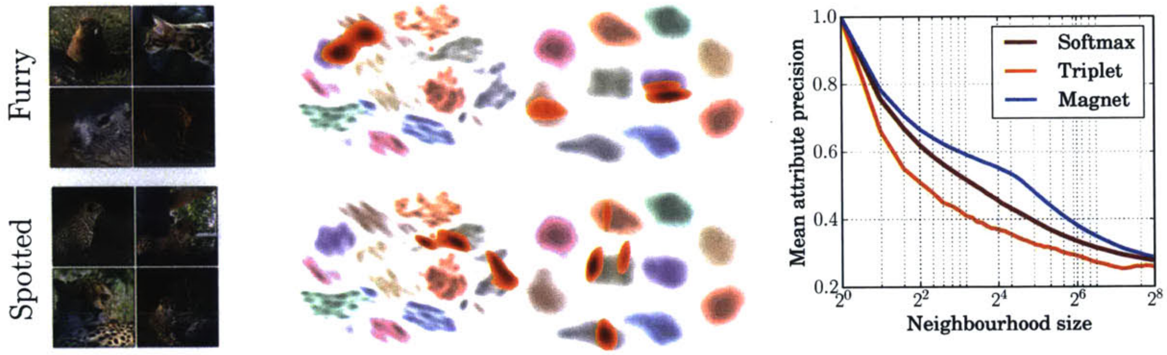
4.4.2 Attribute distribution

We expect Magnet to sculpt a more expressive representation, which enables similar examples of different classes to be close together, and dissimilar examples of the same class to be far apart; this can be seen qualitatively in Figure 4-2. In order to explore this hypothesis quantitatively, after training is complete we examine the attributes of neighbouring examples as a proxy for assessment of similarity. We indeed find the distributions of these attributes to be more concentrated for Magnet.

We attain attribute labels from the Object Attributes dataset (Russakovsky & Fei-Fei, 2010). This provides 25 attribute annotations for 90 classes of an updated version of ImageNet, with about 25 annotated examples per class. Attributes include visual properties such as "striped", "brown", "vegetation" and so on; examples of these can be found in Figure 4-8(a). Annotations are assigned individually for each input, which allows capturing intra-class variation and inter-class invariance.

We train softmax, triplet and Magnet Loss objectives on a curated dataset we refer to as *ImageNet Attributes*. This dataset contains 116,236 examples, and comprises all examples of each of the 90 ImageNet classes for which any attribute annotations are available: in Appendix B.2 we describe it in detail. We emphasize we do not employ any attribute information during training. At convergence, we measure attribute concentration by computing mean attribute precision as function of neighbourhood size. Specifically, for each example and attribute, we compute over different neighbourhood cardinalities the fraction of neighbours also featuring this attribute.

This result can be found in Figure 4-8(d). Magnet Loss outperforms both softmax and triplet losses by a reasonable margin in terms of attribute concentration, with consistent gains of 25-50% over triplet and 10-25% over softmax across neighbourhood sizes. It may seem surprising that softmax surpasses triplet — an approach specifically crafted for distance metric learning. However, note that while the softmax classifier requires high relative projection onto the hyperplane associated with each class, it leaves some flexibility for information retainment in its high-dimensional nullspace. Triplet loss, on the other hand, demands separation based on an imprecise assessment



(a) Attribute examples. (b) Magnet. (c) Softmax. (d) Attribute precision.

Figure 4-8: Attribute concentration properties. **(a)** Examples of images featuring particular attributes. **(b)** & **(c)** The **translucent underlying densities** correspond to the t-SNE visualizations presented in Figure 4-2. These are overlaid with distributions of examples featuring the specified attributes, coloured in **orange**. Magnet clusters together examples of different classes but with similar attributes, whereas softmax and triplet loss (not shown) do not. **(d)** Mean fraction of neighbours featuring the same attributes as function of neighbourhood cardinality. Magnet consistently outperforms softmax and triplet across neighbourhood sizes.

of similarity, resulting in poor proximity of similar examples of different classes.

Magnet's attribute concentration can also be observed visually in Figures 4-8(b) and 4-8(c), presenting the t-SNE projections from Figure 4-2 overlaid with attribute distribution. It can be seen qualitatively that the Magnet attributes are concentrated in particular areas of space, irrespective of class.

4.4.3 Hierarchy recovery

In this experiment, we are interested to see whether each algorithm is able to recover a latent class hierarchy, provided only coarse superclasses. To test this, we randomly pair all classes of ImageNet Attributes, and collapse each pair under a single label. We then train on the corrupted labels, and check whether the finer-grained class labels may be recovered from the learnt representations.

The results can be found in Table 4-4(f). Magnet is able to identify intra-class representation variation, an essential property for success in this task. Softmax also achieves surprisingly competitive results, suggesting that meaningful variation is nevertheless captured within the nullspace of its last layer. For triplet loss, on the other hand, target neighbourhoods are designated prior to training, and as such it is not able to adaptively discriminate finer structure within superclasses.

4.5 Discussion and Remaining Open Problems

In this work, we highlighted a number of difficulties in a class of DML algorithms, and sought to address them. We validated the effectiveness of our approach under a variety of metrics, ranging from classification performance to convergence rate to attribute concentration.

Throughout this work, we anchored in place a number of parameters: we chose the number of clusters K per class as uniform across classes, and refreshed our representation index at a fixed rate. We believe that adaptively varying these during training can enhance performance and facilitate computation.

Another interesting line of work would be to replace the density estimation and indexing component with an approach more sophisticated than K-means. One natural candidate would be a tree-based algorithm. This would enable more efficient and more accurate neighbourhood retrieval.

Chapter 5

Spectral Representations for Convolutional Neural Networks

5.1 Introduction

Convolutional neural networks (CNNs) (LeCun et al., 1989) have been used to achieve unparalleled results across a variety of benchmark machine learning problems, and have been applied successfully throughout science and industry for tasks such as large scale image and video classification (Krizhevsky et al., 2012; Karpathy et al., 2014). One of the primary challenges of CNNs, however, is the computational expense necessary to train them. In particular, the efficient implementation of convolutional kernels has been a key ingredient of any successful use of CNNs at scale.

Due to its efficiency and the potential for amortization of cost, the discrete Fourier transform has long been considered by the deep learning community to be a natural approach to fast convolution (Bengio & LeCun, 2007). More recently, Mathieu et al. (2013); Vasilache et al. (2014) have demonstrated that convolution can be computed significantly faster using discrete Fourier transforms than directly in the spatial domain, even for tiny filters. This computational gain arises from the convenient property of operator duality between convolution in the spatial domain and element-wise multiplication in the frequency domain.

In this work, we argue that the frequency domain offers more than a computational trick for convolution: it also provides a powerful representation for modeling and training CNNs. Frequency decomposition allows studying an input across its various length-scales of variation, and as such provides a natural framework for the analysis of data with spatial coherence. We introduce two applications of spectral representations. These contributions can be applied independently of each other.

SPECTRAL PARAMETRIZATION We propose the idea of learning the filters of CNNs directly in the frequency domain. Namely, we parametrize them as maps of complex numbers, whose discrete Fourier transforms correspond to the usual filter representations in the spatial domain.

Because this mapping corresponds to unitary transformations of the filters, this reparametrization does not alter the underlying model. However, we argue that the spectral representation provides an appropriate domain for parameter optimization, as the frequency basis captures typical filter structure well. More specifically, we show that filters tend to be considerably sparser in their spectral representations, thereby reducing the redundancy that appears in spatial domain representations. This provides the optimizer with more meaningful axis-aligned directions that can be taken advantage of with standard element-wise preconditioning.

We demonstrate the effectiveness of this reparametrization on a number of CNN optimization tasks, converging 2-5 times faster than the standard spatial representation.

SPECTRAL POOLING Pooling refers to dimensionality reduction used in CNNs to impose a capacity bottleneck and facilitate computation. We introduce a new approach to pooling we refer to as *spectral pooling*. It performs dimensionality reduction by projecting onto the frequency basis set and then truncating the representation.

This approach alleviates a number of issues present in existing pooling strategies. For example, while max pooling is featured in almost every CNN and has had great empirical success, one major criticism has been its poor preservation of information (Hinton, 2014b,a). This weakness is exhibited in two ways. First, along with other stride-based pooling approaches, it implies a very sharp dimensionality reduction by at least a factor of 4 every time it is applied on two-dimensional inputs. Moreover, while it encourages translational invariance, it does not utilize its capacity well to reduce approximation loss: the maximum value in each window only reflects very local information, and often does not represent well the contents of the window.

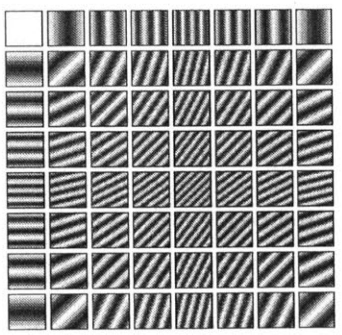
In contrast, we show that spectral pooling preserves considerably more information for the same number of parameters. It achieves this by exploiting the non-uniformity of typical inputs in their signal-to-noise ratio as a function of frequency. For example, natural images are known to have an expected power spectrum that follows an inverse power law: power is heavily concentrated in the lower frequencies — while higher frequencies tend to encode noise (Torralba & Oliva, 2003). As such, the elimination of higher frequencies in spectral pooling not only does minimal damage to the information in the input, but can even be viewed as a type of denoising.

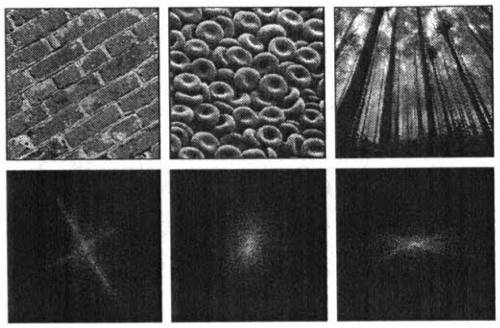
In addition, spectral pooling allows us to specify any arbitrary output map dimensionality. This permits reduction of the map dimensionality in a slow and controlled manner as a function of network depth. Also, since truncation of the frequency representation exactly corresponds to reduction in resolution, we can supplement spectral pooling with stochastic regularization in the form of randomized resolution.

Spectral pooling can be implemented at a negligible additional computational cost in convolutional neural networks that employ FFT for convolution kernels, as it only requires matrix truncation. We also note that these two ideas are both compatible with the recently-introduced method of batch normalization (Ioffe & Szegedy, 2015b), permitting even better training efficiency.

5.2 The Discrete Fourier Transform

The discrete Fourier transform (DFT) is a powerful way to decompose a spatiotemporal signal. In this section, we provide an introduction to a number of components of the DFT drawn upon in this work. We confine ourselves to the two-dimensional DFT, although all properties and results presented can be easily extended to other


(a) DFT basis functions.


(b) Examples of input-transform pairs.

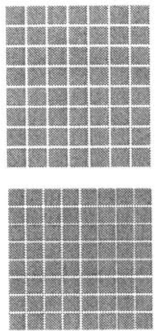

(c) Conj. Symm.

Figure 5-1: Properties of discrete Fourier transforms. **(a)** All discrete Fourier basis functions of map size 8×8. Note the equivalence of some of these due to conjugate symmetry. **(b)** Examples of input images and their frequency representations, presented as log-amplitudes. The frequency maps have been shifted to center the DC component. Rays in the frequency domain correspond to spatial domain edges aligned perpendicular to these. **(c)** Conjugate symmetry patterns for inputs with odd (top) and even (bottom) dimensionalities. **Orange**: real-valuedness constraint. **Blue**: no constraint. **Gray**: value fixed by conjugate symmetry.

input dimensions.

Given an input $\mathbf{x} \in \mathbb{C}^{M \times N}$ (we address the constraint of real inputs in Subsection 5.2.1), its 2D DFT $\mathscr{F}(\mathbf{x}) \in \mathbb{C}^{M \times N}$ is given by

$$\mathscr{F}(\mathbf{x})_{hw} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathrm{x}_{mn} e^{-2\pi i \left(\frac{mh}{M} + \frac{nw}{N}\right)} \qquad \forall h \in \{0, \ldots, M-1\}, \forall w \in \{0, \ldots, N-1\}\ .$$

The DFT is linear and unitary, and so its inverse transform is given by $\mathscr{F}^{-1}(\cdot) = \mathscr{F}(\cdot)^*$, namely the conjugate of the transform itself.

Intuitively, the DFT coefficients resulting from projections onto the different frequencies can be thought of as measures of correlation of the input with basis functions of various length-scales. See Figure 5-1(a) for a visualization of the DFT basis functions, and Figure 5-1(b) for examples of input-frequency map pairs.

The widespread deployment of the DFT can be partially attributed to the development of the Fast Fourier Transform (FFT), a mainstay of signal processing and a standard component of most math libraries. The FFT is an efficient implementation of the DFT with time complexity $\mathcal{O}(MN \log (MN))$.

Convolution using DFT One powerful property of frequency analysis is the operator duality between convolution in the spatial domain and element-wise multiplication in the spectral domain. Namely, given two inputs $\mathbf{x}, \mathbf{f} \in \mathbb{R}^{M \times N}$, we may write

$$\mathscr{F}(\mathbf{x} * \mathbf{f}) = \mathscr{F}(\mathbf{x}) \odot \mathscr{F}(\mathbf{f}) \tag{5.1}$$

where by $*$ we denote a convolution and by $\odot$ an element-wise product.

Approximation error The unitarity of the Fourier basis makes it convenient for the analysis of approximation loss. More specifically, Parseval's Theorem links the ℓ_2 loss between any input $\mathbf{x}$ and its approximation $\hat{\mathbf{x}}$ to the corresponding loss in the frequency domain:

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \|\mathscr{F}(\mathbf{x}) - \mathscr{F}(\hat{\mathbf{x}})\|_2^2 \,. \tag{5.2}$$

An equivalent statement also holds for the inverse DFT operator. This allows us to quickly assess how an input is affected by any distortion we might make to its frequency representation.

5.2.1 Conjugate symmetry constraints

In the following sections of the paper, we will propagate signals and their gradients through DFT and inverse DFT layers. In these layers, we will represent the frequency domain in the complex field. However, for all layers apart from these, we would like to ensure that both the signal and its gradient are constrained to the reals. A necessary and sufficient condition to achieve this is *conjugate symmetry* in the frequency domain. Namely, for any transform $\mathbf{y} = \mathscr{F}(\mathbf{x})$ of some input $\mathbf{x}$, it must hold that

$$\mathbf{y}_{mn} = \mathbf{y}^*_{(M-m)\,\mathrm{mod}M,(N-n)\,\mathrm{mod}N} \qquad \forall m \in \{0, \dots, M-1\}, \forall n \in \{0, \dots, N-1\} \,. \tag{5.3}$$

Thus, intuitively, given the left half of our frequency map, the diminished number of degrees of freedom allows us to reconstruct the right. In effect, this allows us to store approximately half the parameters that would otherwise be necessary. Note, however, that this does not reduce the effective dimensionality, since each element consists of real and imaginary components. The conjugate symmetry constraints are visualized in Figure 5-1(c). Given a real input, its DFT will necessarily meet these. This symmetry can be observed in the frequency representations of the examples in Figure 5-1(b). However, since we seek to optimize over parameters embedded directly in the frequency domain, we need to pay close attention to ensure the conjugate symmetry constraints are enforced upon inversion back to the spatial domain (see Subsection 5.2.2).

5.2.2 Differentiation

Here we discuss how to propagate the gradient through a Fourier transform layer. This analysis can be similarly applied to the inverse DFT layer. Define $\mathbf{x} \in \mathbb{R}^{M\times N}$ and $\mathbf{y} = \mathscr{F}(\mathbf{x})$ to be the input and output of a DFT layer respectively, and $R : \mathbb{R}^{M\times N} \to \mathbb{R}$ a real-valued loss function applied to $\mathbf{y}$ which can be considered as the remainder of the forward pass. Since the DFT is a linear operator, its gradient is simply the transformation matrix itself. During back-propagation, then, this gradient is conjugated, and this, by DFT unitarity, corresponds to the application of the inverse transform:

$$\frac{\partial R}{\partial \mathbf{x}} = \mathscr{F}^{-1}\left(\frac{\partial R}{\partial \mathbf{y}}\right) . \tag{5.4}$$

There is an intricacy that makes matters a bit more complicated. Namely, the conjugate symmetry condition discussed in Subsection 5.2.1 introduces redundancy. Inspecting the conjugate symmetry constraints in Equation (5.3), we note their enforcement of the special case $\mathrm{y}_{00} \in \mathbb{R}$ for N odd, and $\mathrm{y}_{00}, \mathrm{y}_{\frac{N}{2},0}, \mathrm{y}_{0,\frac{N}{2}}, \mathrm{y}_{\frac{N}{2},\frac{N}{2}} \in \mathbb{R}$ for N even. For all other indices they enforce conjugate equality of pairs of distinct elements. These conditions imply that the number of unconstrained parameters is about half the map in its entirety.

5.3 Spectral Pooling

The choice of a pooling technique boils down to the selection of an appropriate set of basis functions to project onto, and some truncation of this representation to establish a lower-dimensionality approximation to the original input. The idea behind spectral pooling stems from the observation that the frequency domain provides an ideal basis for inputs with spatial structure. We first discuss the technical details of this approach, and then its advantages.

Spectral pooling is straightforward to understand and to implement. We assume we are given an input $\mathbf{x} \in \mathbb{R}^{M \times N}$, and some desired output map dimensionality $H \times W$. First, we compute the discrete Fourier transform of the input into the frequency domain as $\mathbf{y} = \mathscr{F}(\mathbf{x}) \in \mathbb{C}^{M \times N}$, and assume that the DC component has been shifted to the center of the domain as is standard practice. We then crop the frequency representation by maintaining only the central $H \times W$ submatrix of frequencies, which we denote as $\hat{\mathbf{y}} \in \mathbb{C}^{H \times W}$. Finally, we map this approximation back into the spatial domain by taking its inverse DFT as $\hat{\mathbf{x}} = \mathscr{F}^{-1}(\hat{\mathbf{y}}) \in \mathbb{R}^{H \times W}$. These steps are listed in Algorithm 1. Note that some of the conjugate symmetry special cases described in Subsection 5.2.2 might be broken by this truncation. As such, to ensure that $\hat{\mathbf{x}}$ is real-valued, we must treat these individually with TREATCORNERCASES, which can be found in the supplementary material.

Figure 5-2 demonstrates the effect of this pooling for various choices of $H \times W$.

Algorithm 1 Spectral pooling

```
Input: Map $\mathbf{x} \in \mathbb{R}^{M \times N}$, output size $H \times W$
Output: Pooled map $\hat{\mathbf{x}} \in \mathbb{R}^{H \times W}$
1: $\mathbf{y} \leftarrow \mathscr{F}(\mathbf{x})$
2: $\hat{\mathbf{y}} \leftarrow$ CROPSPECTRUM($\mathbf{y}, H \times W$)
3: $\hat{\mathbf{y}} \leftarrow$ TREATCORNERCASES($\hat{\mathbf{y}}$)
4: $\hat{\mathbf{x}} \leftarrow \mathscr{F}^{-1}(\hat{\mathbf{y}})$
```

Algorithm 2 Spectral pooling backpropagation

```
Input: Gradient w.r.t output $\frac{\partial R}{\partial \hat{\mathbf{x}}}$
Output: Gradient w.r.t input $\frac{\partial R}{\partial \mathbf{x}}$
1: $\hat{\mathbf{z}} \leftarrow \mathscr{F}\left(\frac{\partial R}{\partial \hat{\mathbf{x}}}\right)$
2: $\hat{\mathbf{z}} \leftarrow$ REMOVEREDUNDANCY($\hat{\mathbf{z}}$)
3: $\mathbf{z} \leftarrow$ PADSPECTRUM($\hat{\mathbf{z}}, M \times N$)
4: $\mathbf{z} \leftarrow$ RECOVERMAP($\mathbf{z}$)
5: $\frac{\partial R}{\partial \mathbf{x}} \leftarrow \mathscr{F}^{-1}(\mathbf{z})$
```

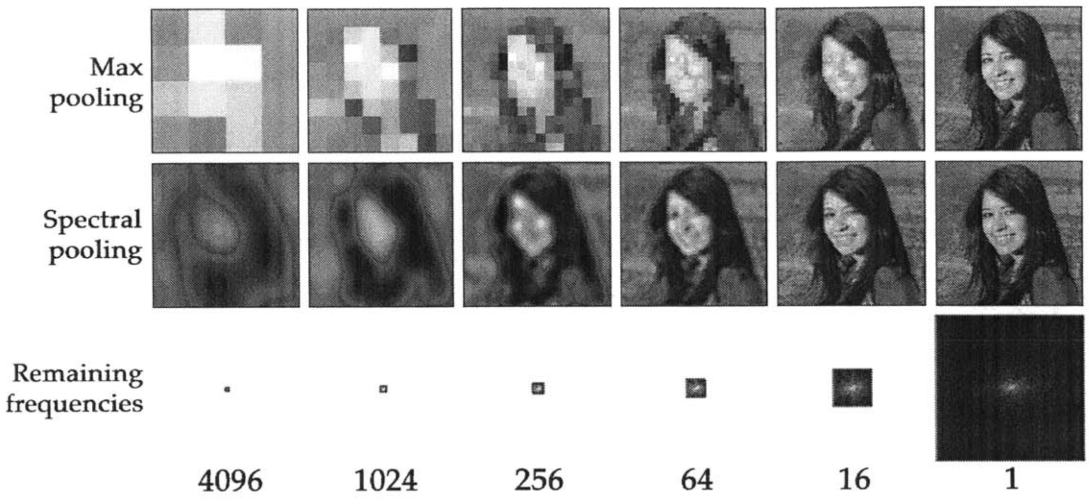


Figure 5-2: Approximations for different pooling schemes, for different factors of dimensionality reduction. Spectral pooling projects onto the Fourier basis and truncates it as desired. This retains significantly more information and permits the selection of any arbitrary output map dimensionality.

The back-propagation procedure is quite intuitive, and can be found in Algorithm 2 (RemoveRedundancy and RecoverMap can be found in the supplementary material). In Subsection 5.2.2, we addressed the nuances of differentiating through DFT and inverse DFT layers. Apart from these, the last component left undiscussed is differentiation through the truncation of the frequency matrix, but this corresponds to a simple zero-padding of the gradient maps to the appropriate dimensions.

In practice, the DFTs are the computational bottlenecks of spectral pooling. However, we note that in convolutional neural networks that employ FFTs for convolution computation, spectral pooling can be implemented at a negligible additional computational cost, since the DFT is performed regardless.

We proceed to discuss a number of properties of spectral pooling, which we then test comprehensively in Section 5.5.

5.3.1 Information preservation

Spectral pooling can significantly increase the amount of retained information relative to max-pooling in two distinct ways. First, its representation maintains more information for the same number of degrees of freedom. Spectral pooling reduces the information capacity by tuning the resolution of the input precisely to match the desired output dimensionality. This operation can also be viewed as linear low-pass filtering and it exploits the non-uniformity of the spectral density of the data with respect to frequency. That is, that the power spectra of inputs with spatial structure, such as natural images, carry most of their mass on lower frequencies. As such, since the amplitudes of the higher frequencies tend to be small, Parseval's theorem from Section 5.2 informs us that their elimination will result in a representation that minimizes the ℓ_2 distortion after reconstruction.

Second, spectral pooling does not suffer from the sharp reduction in output dimensionality exhibited by other pooling techniques. More specifically, for stride-based pooling strategies such as max pooling, the number of degrees of freedom of two-dimensional inputs is reduced by at least 75% as a function of stride. In contrast, spectral pooling allows us to specify any arbitrary output dimensionality, and thus allows us to reduce the map size gradually as a function of layer.

5.3.2 Regularization via resolution corruption

We note that the low-pass filtering radii, say R_H and R_W, can be chosen to be smaller than the output map dimensionalities H, W. Namely, while we truncate our input frequency map to size $H \times W$, we can further zero-out all frequencies outside the central $R_H \times R_W$ square. While this maintains the output dimensionality $H \times W$ of the input domain after applying the inverse DFT, it effectively reduces the resolution of the output. This can be seen in Figure 5-2.

This allows us to introduce regularization in the form of random resolution reduction. We apply this stochastically by assigning a distribution $p_R(\cdot)$ on the frequency truncation radius (for simplicity we apply the same truncation on both axes), sampling from this a random radius at each iteration, and wiping out all frequencies outside the square of that size. Note that this can be regarded as an application of nested dropout (Rippel et al., 2014) on both dimensions of the frequency decomposition of our input. In practice, we have had success choosing $p_R(\cdot) = U_{[H_{\min}, H]}(\cdot)$, i.e., a uniform distribution stretching from some minimum value all the way up to the highest possible resolution.

5.4 Spectral Parametrization of CNNs

Here we demonstrate how to learn the filters of CNNs directly in their frequency domain representations. This offers significant advantages over the traditional spatial representation, which we show empirically in Section 5.5.

Let us assume that for some layer of our convolutional neural network we seek to learn filters of size $H \times W$. To do this, we parametrize each filter $\mathbf{f} \in \mathbb{C}^{H \times W}$ in our network directly in the frequency domain. To attain its spatial representation, we simply compute its inverse DFT as $\mathscr{F}^{-1}(\mathbf{f}) \in \mathbb{R}^{H \times W}$. From this point on, we proceed as we would for any standard CNN by computing the convolution of the filter with inputs in our minibatch, and so on.

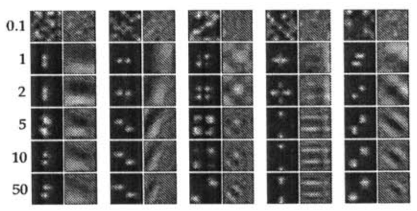
The back-propagation through the inverse DFT is virtually identical to the one of spectral pooling described in Section 5.3. We compute the gradient as outlined in Subsection 5.2.2, being careful to obey the conjugate symmetry constraints discussed in Subsection 5.2.1.

We emphasize that this approach does not change the underlying CNN model in any way — only the way in which it is parametrized. Hence, this only affects the way the solution space is explored by the optimization procedure.

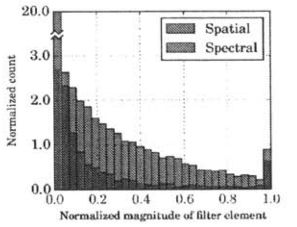
5.4.1 Leveraging filter structure

This idea exploits the observation that CNN filters have a very characteristic structure that reappears across data sets and problem domains. That is, CNN weights can typically be captured with a small number of degrees of freedom. Represented in the spatial domain, however, this results in significant redundancy.

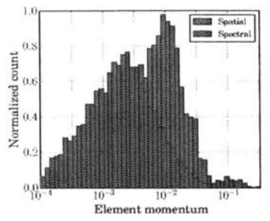
The frequency domain, on the other hand, provides an appealing basis for filter representation: characteristic filters (e.g., Gabor filters) are often very localized in their spectral representations. This follows from the observation that filters tend to



(a) Filters over time.



(b) Sparsity patterns.



(c) Momenta distributions.

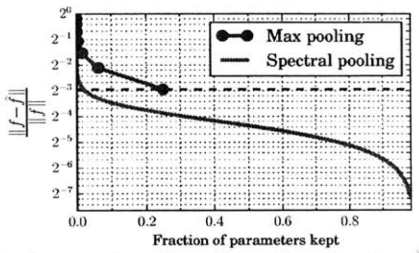
Figure 5-3: Learning dynamics of CNNs with spectral parametrization. The histograms have been produced after 10 epochs of training on CIFAR-10 by each method, but are similar throughout. **(a)** Progression over several epochs of filters parametrized in the frequency domain. Each pair of columns corresponds to the spectral parametrization of a filter and its inverse transform to the spatial domain. Filter representations tend to be more local in the Fourier basis. **(b)** Sparsity patterns for the different parametrizations. Spectral representations tend to be considerably sparser. **(c)** Distributions of momenta across parameters for CNNs trained with and without spectral parametrization. In the spectral parametrization considerably fewer parameters are updated.

feature very specific length-scales and orientations. Hence, they tend to have nonzero support in a narrow set of frequency components. This hypothesis can be observed qualitatively in Figure 5-3(a) and quantitatively in Figure 5-3(b).

Empirically, in Section 5.5 we observe that spectral representations of filters leads to a convergence speedup by 2-5 times. We remark that, had we trained our network with standard stochastic gradient descent, the linearity of differentiation and parameter update would have resulted in exactly the same filters regardless of whether they were represented in the spatial or frequency domain during training (this is true for *any* invertible linear transformation of the parameter space).

However, as discussed, this parametrization corresponds to a rotation to a more meaningful axis alignment, where the number of relevant elements has been significantly reduced. Since modern optimizers implement update rules that consist of adaptive element-wise rescaling, they are able to leverage this axis alignment by making large updates to a small number of elements. This can be seen quantitatively in Figure 5-3(c), where the optimizer — Adam (Kingma & Ba, 2015), in this case — only touches a small number of elements in its updates.

There exist a number of extensions of the above approach we believe would be



(a) Approximation loss for the ImageNet validation set.

Method	**CIFAR-10**	**CIFAR-100**
Stochastic pooling	15.13%	41.51%
Maxout	11.68%	38.57%
Network-in-network	10.41%	35.68%
Deeply supervised	9.78%	34.57%
Spectral pooling	**8.6%**	**31.6%**

(b) Classification rates.

Figure 5-4: **(a)** Average information dissipation for the ImageNet validation set as a function of fraction of parameters kept. This is measured in ℓ_2 error normalized by the input norm. The red horizontal line indicates the best error rate achievable by max pooling. **(b)** Test errors on CIFAR-10/100 without data augmentation of the optimal spectral pooling architecture, as compared to current state-of-the-art approaches: stochastic pooling (Zeiler & Fergus, 2013), Maxout (Goodfellow et al., 2013), network-in-network (Lin et al., 2013), and deeply-supervised nets (Lee et al., 2014).

quite promising in future work; we elaborate on these in the discussion.

5.5 Experiments

We demonstrate the effectiveness of spectral representations in a number of different experiments. We ran all experiments on code optimized for the Xeon Phi coprocessor. We used Spearmint (Snoek et al., 2015) for Bayesian optimization of hyperparameters with 5-20 concurrent evaluations.

5.5.1 Spectral pooling

Information preservation We test the information retainment properties of spectral pooling on the validation set of ImageNet (Russakovsky et al., 2015). For the different pooling strategies we plot the average approximation loss resulting from pooling to different dimensionalities. This can be seen in Figure 5-4. We observe the two aspects discussed in Subsection 5.3.1: first, spectral pooling permits significantly better reconstruction for the same number of parameters. Second, for max pooling, the only knob controlling the coarseness of approximation is the stride, which results in severe quantization and a constraining lower bound on preserved information (marked in the figure as a horizontal red line). In contrast, spectral pooling permits the selection of any output dimensionality, thereby producing a smooth curve over all frequency truncation choices.

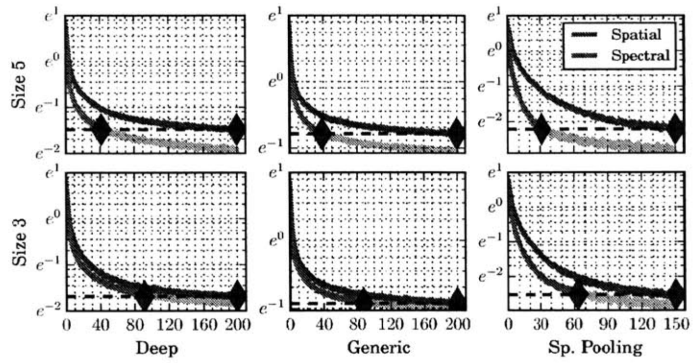
CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS We test spectral pooling on different classification tasks. We hyperparametrize and optimize the following CNN architecture:

$$\left(\mathrm{C}_{3\times3}^{96+32m} \rightarrow \mathrm{SP}_{\downarrow\lfloor\gamma H_m\rfloor\times\lfloor\gamma H_m\rfloor}\right)_{m=1}^{M} \rightarrow \mathrm{C}_{1\times1}^{96+32M} \rightarrow \mathrm{C}_{1\times1}^{10/100} \rightarrow \mathrm{GA} \rightarrow \mathrm{Softmax} \tag{5.5}$$

Here, by C_S^F we denote a convolutional layer with F filters each of size S, by $\mathrm{SP}_{\downarrow S}$ a spectral pooling layer with output dimensionality S, and GA the global averaging layer described in Lin et al. (2013). We upper-bound the number of filters per layer as 288. Every convolution and pooling layer is followed by a ReLU nonlinearity. We let H_m be the height of the map of layer m. Hence, each spectral pooling layer reduces each output map dimension by factor $\gamma \in (0, 1)$. We assign frequency dropout distribution $p_R(\cdot; m, \alpha, \beta) = U_{[\lfloor c_m H_m\rfloor, H_m]}(\cdot)$ for layer m, total layers M and with $c_m(\alpha, \beta) = \alpha + \frac{m}{M}(\beta - \alpha)$ for some constants $\alpha, \beta \in \mathbb{R}$. This parametrization can be thought of as some linear parametrization of the dropout rate as a function of the layer.

We perform hyperparameter optimization on the dimensionality decay rate $\gamma \in [0.25, 0.85]$, number of layers $M \in \{1, \ldots, 15\}$, resolution randomization hyperparameters $\alpha, \beta \in [0, 0.8]$, weight decay rate in $[10^{-5}, 10^{-2}]$, momentum in $[1-0.1^{0.5}, 1-0.1^{2}]$ and initial learning rate in $[0.1^{4}, 0.1]$. We train each model for 150 epochs and anneal the learning rate by a factor of 10 at epochs 100 and 140. We intentionally use no dropout nor data augmentation, as these introduce a number of additional hyperparameters which we want to disambiguate as alternative factors for success.

Perhaps unsurprisingly, the optimal hyperparameter configuration assigns the



(a) Training curves.

Architecture	Filter size	Speedup factor
Deep (5.7)	3×3	2.2
Deep (5.7)	5×5	4.8
Generic (5.6)	3×3	2.2
Generic (5.6)	5×5	5.1
Sp. Pooling (5.5)	3×3	2.4
Sp. Pooling (5.5)	5×5	4.8

(b) Speedup factors.

Figure 5-5: Optimization of CNNs via spectral parametrization. All experiments include data augmentation. **(a)** Training curves for the various experiments. The remainder of the optimization past the matching point is marked in light blue. The red diamonds indicate the relative epochs in which the asymptotic error rate of the spatial approach is achieved. **(b)** Speedup factors for different architectures and filter sizes. A non-negligible speedup is observed even for tiny 3×3 filters.

slowest possible layer map decay rate $\gamma = 0.85$. It selects randomized resolution reduction constants of about $\alpha \approx 0.30, \beta \approx 0.15$, momentum of about 0.95 and initial learning rate 0.0088. These settings allow us to attain classification rates of 8.6% on CIFAR-10 and 31.6% on CIFAR-100. These are competitive results among approaches that do not employ data augmentation: a comparison to state-of-the-art approaches from the literature can be found in Table 5-4(b).

5.5.2 Spectral parametrization of CNNs

We demonstrate the effectiveness of spectral parametrization on a number of CNN optimization tasks, for different architectures and for different filter sizes. We use the notation MP_S^T to denote a max pooling layer with size S and stride T, and FC^F is a fully-connected layer with F filters.

The first architecture is the generic one used in a variety of deep learning papers, such as Krizhevsky et al. (2012); Snoek et al. (2012); Krizhevsky (2009); Kingma &

Ba (2015):

$$C^{96}_{3\times3} \rightarrow MP^{2}_{3\times3} \rightarrow C^{192}_{3\times3} \rightarrow MP^{2}_{3\times3} \rightarrow FC^{1024} \rightarrow FC^{512} \rightarrow \text{Softmax} \tag{5.6}$$

The second architecture we consider is the one employed in Snoek et al. (2015), which was shown to attain competitive classification rates. It is deeper and more complex:

$$C^{96}_{3\times3} \rightarrow C^{96}_{3\times3} \rightarrow MP^{2}_{3\times3} \rightarrow C^{192}_{3\times3} \rightarrow C^{192}_{3\times3} \rightarrow C^{192}_{3\times3} \rightarrow MP^{2}_{3\times3} \rightarrow C^{192}_{1\times1} \rightarrow C^{10/100}_{1\times1} \rightarrow GA \rightarrow \text{Softmax} \tag{5.7}$$

The third architecture considered is the spectral pooling network from Equation 5.5. To increase the difficulty of optimization and reflect real training conditions, we supplemented all networks with considerable data augmentation in the form of translations, horizontal reflections, HSV perturbations and dropout.

We initialized both spatial and spectral filters in the spatial domain as the same values; for the spectral parametrization experiments we then computed the Fourier transform of these to attain their frequency representations. We optimized all networks using the Adam (Kingma & Ba, 2015) update rule, a variant of RMSprop that we find to be a fast and robust optimizer.

The training curves can be found in Figure 5-5(a) and the respective factors of convergence speedup in Table 5-5. Surprisingly, we observe non-negligible speedup even for tiny filters of size 3×3, where we did not expect the frequency representation to have much room to exploit spatial structure.

5.6 Discussion and Remaining Open Problems

In this work, we demonstrated that spectral representations provide a rich spectrum of applications. We introduced spectral pooling, which allows pooling to any desired output dimensionality while retaining significantly more information than other pooling approaches. In addition, we showed that the Fourier functions provide a suitable basis for filter parametrization, as demonstrated by faster convergence of the optimization procedure.

One possible future line of work is to embed the network in its entirety in the frequency domain. In models that employ Fourier transforms to compute convolutions, at every convolutional layer the input is FFT-ed and the element-wise multiplication output is then inverse FFT-ed. These back-and-forth transformations are very computationally intensive, and as such it would be desirable to strictly remain in the frequency domain. However, the reason for these repeated transformations is the application of nonlinearities in the forward domain: if one were to propose a sensible nonlinearity in the frequency domain, this would spare us from the incessant domain switching.

In addition, one significant downfall of the DFT approach is its difficulty in handling finite impulse response filtering. In particular, its projection onto the various frequencies involves global sums over the entire input. Hence, the input domain has perfect spatial locality and no spectral locality, while the Fourier domain has perfect spectral locality and no spatial locality. An intermediate solution we believe would be very effective is employing wavelets, which provide a middle ground between the two approaches. While wavelets have been employed throughout machine learning with great promise (Bruna & Mallat, 2013; Oyallon et al., 2013), to our knowledge they have not been used in an adaptive way to learn CNNs.

Chapter 6

Scalable Bayesian Optimization Using Deep Neural Networks

6.1 Introduction

In the last few years, the field of machine learning has seen unprecedented growth due to a new wealth of data, increases in computational power, new algorithms, and a plethora of exciting new applications. As researchers tackle more ambitious problems, the models they use are also becoming more sophisticated. However, the growing complexity of machine learning models inevitably comes with the introduction of additional hyperparameters. These range from design decisions such as the shape of a neural network architecture, to optimization parameters such as learning rates, to regularization hyperparameters such as weight decay. Proper setting of these hyperparameters is critical for performance on difficult problems.

There are many methods for optimizing over hyperparameter settings, ranging from simplistic procedures like grid or random search Bergstra & Bengio (2012), to more sophisticated model-based approaches using random forests Hutter et al. (2011) or Gaussian processes Snoek et al. (2012). Bayesian optimization is a natural framework for model-based global optimization of noisy, expensive black-box functions. It offers a principled approach to modeling uncertainty, which allows exploration and exploitation to be naturally balanced during the search. Perhaps the most commonly used model for Bayesian optimization is the Gaussian process (GP) due to its simplicity and flexibility in terms of conditioning and inference.

However, a major drawback of GP-based Bayesian optimization is that inference time grows cubically in the number of observations, as it necessitates the inversion of a dense covariance matrix. For problems with a very small number of hyperparameters, this has not been an issue, as the minimum is often discovered before the cubic scaling renders further evaluations prohibitive. As the complexity of machine learning models grows, however, the size of the search space grows as well, along with the number of hyperparameter configurations that need to be evaluated before a solution of sufficient quality is found. Fortunately, as models have grown in complexity, computation has become significantly more accessible and it is now possible to train many models in parallel. A natural solution to the hyperparameter search problem is to therefore combine large-scale parallelism with a scalable Bayesian optimization method. The cubic scaling of the GP, however, has made it infeasible to pursue this approach.

The goal of this work is to develop a method for scaling Bayesian optimization, while still maintaining its desirable flexibility and characterization of uncertainty. To that end, we propose the use of neural networks to learn an adaptive set of basis functions for Bayesian linear regression. We refer to this approach as Deep Networks for Global Optimization (DNGO). Unlike a standard Gaussian process, DNGO scales linearly with the number of function evaluations—which, in the case of hyperparameter optimization, corresponds to the number of models trained—and is amenable to stochastic gradient training. Although it may seem that we are merely moving the problem of setting the hyperparameters of the model being tuned to setting them for the tuner itself, we show that for a suitable set of design choices it is possible to create a robust, scalable, and effective Bayesian optimization system that generalizes across many global optimization problems.

We demonstrate the effectiveness of DNGO on a number of difficult problems, including benchmark problems for Bayesian optimization, convolutional neural networks for object recognition, and multi-modal neural language models for image caption generation. We find hyperparameter settings that achieve competitive with state-of-the-art results of 6.37% and 27.4% on CIFAR-10 and CIFAR-100 respectively, and BLEU scores of 25.1 and 26.7 on the Microsoft COCO 2014 dataset using a single model and a 3-model ensemble.

6.2 Background and Related Work

6.2.1 Bayesian Optimization

Bayesian optimization is a well-established strategy for the global optimization of noisy, expensive black-box functions Mockus et al. (1978). For an in-depth review, see Lizotte (2008), Brochu et al. (2010) and Osborne et al. (2009). Bayesian optimization relies on the construction of a probabilistic model that defines a distribution over objective functions from the input space to the objective of interest. Conditioned on a prior over the functional form and a set of N observations of input-target pairs $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, the relatively cheap posterior over functions is then queried to reason about where to seek the optimum of the expensive function of interest. The promise of a new experiment is quantified using an *acquisition function*, which, applied to the posterior mean and variance, expresses a trade-off between exploration and exploitation. Bayesian optimization proceeds by performing a proxy optimization over this acquisition function in order to determine the next input to evaluate.

Recent innovation has resulted in significant progress in Bayesian optimization, including elegant theoretical results Srinivas et al. (2010); Bull (2011); de Freitas et al. (2012), multitask and transfer optimization Krause & Ong (2011); Swersky et al. (2013); Bardenet et al. (2013) and the application to diverse tasks such as sensor set selection Garnett et al. (2010), the tuning of adaptive Monte Carlo Mahendran et al. (2012) and robotic gait control Calandra et al. (2014b).

Typically, GPs have been used to construct the distribution over functions used in Bayesian optimization, due to their flexibility, well-calibrated uncertainty, and analytic properties Jones (2001); Osborne et al. (2009). Recent work has sought to improve the performance of the GP approach through accommodating higher dimensional problems Wang et al. (2013); Djolonga et al. (2013), input non-stationarities Snoek et al. (2014) and initialization through meta-learning Feurer et al. (2015). Random forests, which scale linearly with the data, have also been used successfully for algorithm configuration by Hutter et al. (2011) with empirical estimates of model uncertainty.

More specifically, Bayesian optimization seeks to solve the minimization problem

$$\mathbf{x}^\star = \underset{\mathbf{x}\in\mathcal{X}}{\arg\min}\, f(\mathbf{x})\,, \tag{6.1}$$

where we take $\mathcal{X}$ to be a compact subset of $\mathbb{R}^K$. In our work, we build upon the standard GP-based approach of Jones (2001) which uses a GP surrogate and the *expected improvement* acquisition function Mockus et al. (1978). For the surrogate model hyperparameters $\boldsymbol{\Theta}$, let $\sigma^2(\mathbf{x};\boldsymbol{\Theta}) = \Sigma(\mathbf{x},\mathbf{x};\boldsymbol{\Theta})$ be the marginal predictive variance of the probabilistic model, $\mu(\mathbf{x};\mathcal{D},\boldsymbol{\Theta})$ be the predictive mean, and define

$$\gamma(\mathbf{x}) = \frac{f(\mathbf{x}_{\text{best}}) - \mu(\mathbf{x};\mathcal{D},\boldsymbol{\Theta})}{\sigma(\mathbf{x};\mathcal{D},\boldsymbol{\Theta})}\,, \tag{6.2}$$

where $f(\mathbf{x}_{\text{best}})$ is the lowest observed value. The expected improvement criterion is defined as

$$\begin{aligned} a_{\text{EI}}(\mathbf{x};\mathcal{D},\boldsymbol{\Theta}) = & \\ & \sigma(\mathbf{x};\mathcal{D},\boldsymbol{\Theta})\left[\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x});0,1)\right]\,. \end{aligned} \tag{6.3}$$

Here $\Phi(\cdot)$ is the cumulative distribution function of a standard normal, and $\mathcal{N}(\cdot;0,1)$ is the density of a standard normal. Note that numerous alternate acquisition functions and combinations thereof have been proposed Kushner (1964); Srinivas et al. (2010); Hoffman et al. (2011), which could be used without affecting the analytic properties of our approach.

6.2.2 Bayesian Neural Networks

The idea of applying Bayesian methods to neural networks has a rich history in machine learning MacKay (1992); Hinton & van Camp (1993); Buntine & Weigend (1991); Neal (1995); De Freitas (2003). The goal of Bayesian neural networks is to uncover the full posterior distribution over the network weights in order to capture uncertainty, to act as a regularizer, and to provide a framework for model comparison. The full posterior is, however, intractable for most forms of neural networks, necessitating expensive approximate inference or Markov chain Monte Carlo simulation. More recently, full or approximate Bayesian inference has been considered for small pieces of the overall architecture. For example, in similar spirit to this work, Lázaro-Gredilla & Figueiras-Vidal (2010); Hinton & Salakhutdinov (2008) and Calandra et al. (2014a) considered inference over just the last layer of a neural network. Other examples can be found in Kingma & Welling (2014); Rezende et al. (2014) and Mnih & Gregor (2014), where a neural network is used in a variational approximation to the posterior distribution over the latent variables of a directed generative neural network.

6.3 Adaptive Basis Regression with Deep Neural Networks

A key limitation of GP-based Bayesian optimization is that the computational cost of the technique scales cubically in the number of observations, limiting the applicability of the approach to objectives that require a relatively small number of observations to optimize. In this work, we aim to replace the GP traditionally used in Bayesian optimization with a model that scales in a less dramatic fashion, but retains most of the GP's desirable properties such as flexibility and well-calibrated uncertainty. Bayesian neural networks are a natural consideration, not least because of the theoretical relationship between Gaussian processes and infinite Bayesian neural networks Neal (1995); Williams (1996). However, deploying these at a large scale is very computationally expensive.

As such, we take a pragmatic approach and add a Bayesian linear regressor to the last hidden layer of a deep neural network, marginalizing only the output weights of the net while using a point estimate for the remaining parameters. This results in *adaptive basis regression*, a well-established statistical technique which scales linearly in the number of observations, and cubically in the basis function dimensionality. This allows us to explicitly trade off evaluation time and model capacity. As such, we form the basis using the very flexible and powerful non-linear functions defined by the neural network.

First of all, without loss of generality and assuming compact support for each input dimension, we scale the input space to the unit hypercube. We denote by $\boldsymbol{\phi}(\cdot) = [\phi_1(\cdot), \ldots, \phi_D(\cdot)]^\mathsf{T}$ the vector of outputs from the last hidden layer of the network, trained on inputs and targets $\mathcal{D} := \{(\mathbf{x}_n, y_n)\}_{n=1}^N \subset \mathbb{R}^K \times \mathbb{R}$. We take these to be our set of basis functions. In addition, define $\mathbf{\Phi}$ to be the design matrix arising from the data and this basis, where $\Phi_{nd} = \phi_d(\mathbf{x}_n)$ is the output design matrix, and $\mathbf{y}$ the stacked target vector.

These basis functions are parameterized via the weights and biases of the deep neural network, and these parameters are trained via backpropagation and stochastic gradient descent with momentum. In this training phase, a linear output layer is also fit. This procedure can be viewed as a maximum *a posteriori* (MAP) estimate of all parameters in the network. Once this "basis function neural network" has been trained, we replace the MAP-parameterized output layer with a Bayesian linear regressor that captures uncertainty in the weights. See Section 6.3.1 for a more elaborate explanation of this choice.

The predictive mean $\mu(\mathbf{x}; \Theta)$ and variance $\sigma^2(\mathbf{x}; \Theta)$ of the model are then given by (see Bishop, 2006)

$$\mu(\mathbf{x}; \mathcal{D}, \Theta) = \mathbf{m}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) + \eta(\mathbf{x}) \,, \tag{6.4}$$

$$\sigma^2(\mathbf{x}; \mathcal{D}, \Theta) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\mathbf{K}^{-1}\boldsymbol{\phi}(\mathbf{x}) + \frac{1}{\beta} \tag{6.5}$$

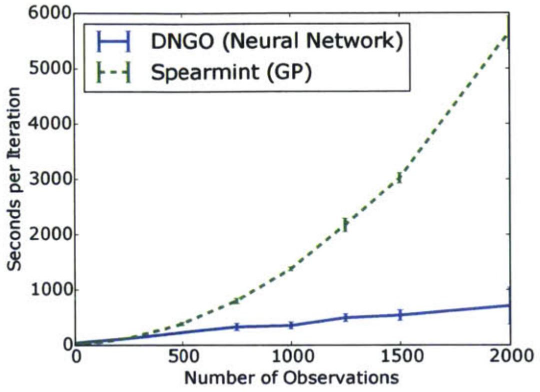


Figure 6-1: A comparison of the time per suggested experiment for our method compared to the state-of-the-art GP based approach Snoek et al. (2014) on the six dimensional Hartmann function. We ran each algorithm on the same 32 core system with 80GB of RAM five times and plot the mean and standard deviation.

where

$$\mathbf{m} = \beta \mathbf{K}^{-1} \mathbf{\Phi}^T \tilde{\mathbf{y}} \in \mathbb{R}^D \tag{6.6}$$

$$\mathbf{K} = \beta \mathbf{\Phi}^T \mathbf{\Phi} + \boldsymbol{I}\alpha \in \mathbb{R}^{D \times D}. \tag{6.7}$$

Here, $\eta(\mathbf{x})$ is a prior mean function which is described in Section 6.3.1, and $\tilde{\mathbf{y}} = \mathbf{y} - \eta(\mathbf{x})$. In addition, $\alpha, \beta \in \mathbf{\Theta}$ are regression model hyperparameters. We integrate out α and β using slice sampling Neal (2000) according to the methodology of Snoek et al. (2012) over the marginal likelihood, which is given by

$$\begin{aligned}\log p(\mathbf{y} \mid \mathbf{X}, \alpha, \beta) = \frac{D}{2} \log \alpha + \frac{N}{2} \log \beta - \frac{N}{2} \log(2\pi) \\ - \frac{\beta}{2} (\tilde{\mathbf{y}} - \Phi \mathbf{m})^2 - \frac{\alpha}{2} \mathbf{m}^\mathsf{T} \mathbf{m} - \frac{1}{2} \log |\mathbf{K}| \,.\end{aligned} \tag{6.8}$$

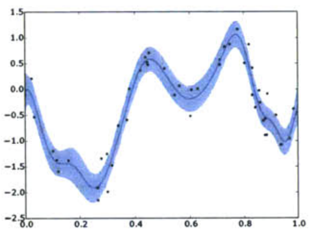
It is clear that the computational bottleneck of this procedure is the inversion of $\mathbf{K}$. However, note that the size of this matrix grows with the output dimensionality D, rather than the number of observations N as in the GP case. This allows us to scale to significantly more observations than with the GP as demonstrated in Figure 6-1.

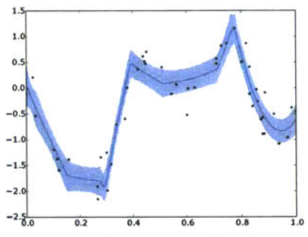
6.3.1 Model details

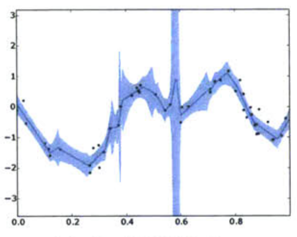
Network architecture

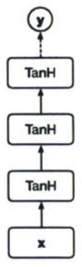
A natural concern with the use of deep networks is that they often require significant effort to tune and tailor to specific problems. One can consider adjusting the architecture and tuning the hyperparameters of the neural network as itself a difficult hyperparameter optimization problem. An additional challenge is that we aim to create an approach that generalizes across optimization problems. We found that design decisions such as the type of activation function used significantly altered the performance of the Bayesian optimization routine. For example, in Figure **6-2** we see that the commonly used rectified linear (ReLU) function can lead to very poor estimates of uncertainty, which causes the Bayesian optimization routine to explore excessively. Since the bounded tanh function results in smooth functions with realistic variance, we use this nonlinearity in this work; however, if the smoothness assumption needs to be relaxed, a combination of rectified linear functions with a tanh function only on the last layer can also be used in order to bound the basis.

In order to tune any remaining hyperparameters, such as the width of the hidden layers and the amount of regularization, we used GP-based Bayesian optimization. For each of one to four layers we ran Bayesian optimization using the Spearmint **Snoek et al. (2014)** package to minimize the average relative loss on a series of benchmark global optimization problems. We tuned a global learning rate, momentum, layer sizes, ℓ_2 normalization penalties for each set of weights and dropout rates **Hinton et al. (2012b)** for each layer. Interestingly, the optimal configuration featured no dropout and very modest ℓ_2 normalization. We suspect that dropout, despite having an approximate correction term, causes noise in the predicted mean resulting in a loss of precision. The optimizer instead preferred to restrict capacity via a small number of hidden units. Namely, the optimal architecture is a deep and narrow network with 3 hidden layers and approximately 50 hidden units per layer. We use the same architecture throughout all of our empirical evaluation, and this architecture is illustrated in Figure **6-2(d)**.


(a) TanH Units


(b) ReLU + TanH Units


(c) ReLU Units



(d)

Figure 6-2: A comparison of the predictive mean and uncertainty learned by the model when using 6-2(a) only tanh, 6-2(c) only rectified linear (ReLU) activation functions or 6-2(b) ReLU's but a tanh on the last hidden layer. The shaded regions correspond to standard deviation envelopes around the mean. The choice of activation function significantly modifies the basis functions learned by the model. Although the ReLU, which is the standard for deep neural networks, is highly flexible, we found that its unbounded activation can lead to extremely large uncertainty estimates. Subfigure 6-2(d) illustrates the overall architecture of the DNGO model. Dashed lines correspond to weights that are marginalized.

Marginal likelihood vs MAP estimate

The standard empirical Bayesian approach to adaptive basis regression is to maximize the marginal likelihood with respect to the parameters of the basis (see Equation 6.8), thus taking the model uncertainty into account. However, in the context of our method, this requires evaluating the gradient of the marginal likelihood, which requires inverting a $D \times D$ matrix on each update of stochastic gradient descent. As this makes the optimization of the net significantly slower, we take a pragmatic approach and optimize the basis using a point estimate and apply the Bayesian linear regression layer *post-hoc*. We found that both approaches gave qualitatively and empirically similar results, and as such we in practice employ the more efficient one.

Quadratic Prior

One of the advantages of Bayesian optimization is that it provides natural avenues for incorporating prior information about the objective function and search space. For example, when choosing the boundaries of the search space, a typical assumption has been that the optimal solution lies somewhere in the interior of the input space.

Experiment	# Evals	SMAC	TPE	Spearmint	DNGO
Branin (0.398)	200	0.655 ± 0.27	0.526 ± 0.13	$\mathbf{0.398 \pm 0.00}$	$\mathbf{0.398 \pm 0.00}$
Hartmann6 (-3.322)	200	-2.977 ± 0.11	-2.823 ± 0.18	$\mathbf{-3.3166 \pm 0.02}$	-3.319 ± 0.00
Logistic Regression	100	8.6 ± 0.9	8.2 ± 0.6	$\mathbf{6.88 \pm 0.0}$	$\mathbf{6.89 \pm 0.04}$
LDA (On grid)	50	1269.6 ± 2.9	1271.5 ± 3.5	$\mathbf{1266.2 \pm 0.1}$	$\mathbf{1266.2 \pm 0.0}$
SVM (On grid)	100	$\mathbf{24.1 \pm 0.1}$	24.2 ± 0.0	$\mathbf{24.1 \pm 0.1}$	$\mathbf{24.1 \pm 0.1}$

Table 6.1: Evaluation of DNGO on global optimization benchmark problems versus scalable (TPE, SMAC) and non-scalable (Spearmint) Bayesian optimization methods. All problems are minimization problems. For each problem, each method was run 10 times to produce error bars.

However, by the curse of dimensionality, most of the volume of the space lies very close to its boundaries. Therefore, we select a mean function $\eta(\mathbf{x})$ (see Equation 6.4) to reflect our subjective prior beliefs that the function is coarsely approximated by a convex quadratic function centered in the bounded search region, i.e.,

$$\eta(\mathbf{x}) = \lambda + (\mathbf{x} - \mathbf{c})^T \mathbf{\Lambda} (\mathbf{x} - \mathbf{c}) \tag{6.9}$$

where $\mathbf{c}$ is the center of the quadratic, λ is an offset and $\mathbf{\Lambda}$ a diagonal scaling matrix. We place a Gaussian prior with mean 0.5 (the center of the unit hypercube) on $\mathbf{c}$, horseshoe Carvalho et al. (2009) priors on the diagonal elements $\Lambda_{kk}\ \forall k \in \{1, \ldots, K\}$ and integrate out b, λ and $\mathbf{c}$ using slice sampling over the marginal likelihood.

The horseshoe is a so-called *one-group* prior for inducing sparsity and is a somewhat unusual choice for the weights of a regression model. Here we choose it because it 1) has support only on the positive reals, leading to convex functions, and 2) it has a large spike at zero with a heavy tail, resulting in strong shrinkage for small values while preserving large ones. This last effect is important for handling model misspecification as it allows the quadratic effect to disappear and become a simple offset if necessary.

6.3.2 Incorporating input space constraints

Many problems of interest have complex, possibly unknown bounds, or exhibit undefined behavior in some regions of the input space. These regions can be characterized as *constraints* on the search space. Recent work Gelbart et al. (2014); Snoek (2013); Gramacy & Lee (2010) has developed approaches for modeling unknown constraints in GP-based Bayesian optimization by learning a constraint classifier and then discounting expected improvement by the probability of constraint violation.

More specifically, define $c_n \in \{0, 1\}$ to be a binary indicator of the validity of input $\mathbf{x}_n$. Also, denote the sets of valid and invalid inputs as $\mathcal{V} = \{(\mathbf{x}_n, y_n) \mid c_n = 1\}$ and $\mathcal{I} = \{(\mathbf{x}_n, y_n) \mid c_n = 0\}$, respectively. Note that $\mathcal{D} := \mathcal{V} \cup \mathcal{I}$. Lastly, let $\mathbf{\Psi}$ be the collection of constraint hyperparameters. The modified expected improvement function can be written as

$$a_{\mathrm{CEI}}(\mathbf{x}; \mathcal{D}, \mathbf{\Theta}, \mathbf{\Psi}) = a_{\mathrm{EI}}(\mathbf{x}; \mathcal{V}, \mathbf{\Theta}) \mathbb{P}\left[c = 1 \mid \mathbf{x}, \mathcal{D}, \mathbf{\Psi}\right] .$$

In this work, to model the constraint surface, we similarly replace the Gaussian process with the adaptive basis model, integrating out the output layer weights:

$$\begin{aligned} &\mathbb{P}[c = 1 \mid \mathbf{x}, \mathcal{D}, \mathbf{\Psi}] = \\ &\qquad \int_{\mathbf{w}} \mathbb{P}\left[c = 1 \mid \mathbf{x}, \mathcal{D}, \mathbf{w}, \mathbf{\Psi}\right] \mathrm{P}(\mathbf{w}; \mathbf{\Psi}) \mathrm{d}\mathbf{w} . \end{aligned} \tag{6.10}$$

In this case, we use a Laplace approximation to the posterior. For noisy constraints we perform Bayesian logistic regression, using a logistic likelihood function for $\mathbb{P}\left[c = 1 \mid \mathbf{x}, \mathcal{D}, \mathbf{w}, \mathbf{\Psi}\right]$. For noiseless constraints, we replace the logistic function with a step function.

6.3.3 Parallel Bayesian Optimization

Obtaining a closed form expression for the joint acquisition function across multiple inputs is intractable in general Ginsbourger & Riche (2010). However, a successful Monte Carlo strategy for parallelizing Bayesian optimization was developed in Snoek et al. (2012). The idea is to marginalize over the possible outcomes of currently running experiments when making a decision about a new experiment to run. Following this strategy, we use the posterior predictive distribution given by Equations 6.4 and 6.5 to generate a set of fantasy outcomes for each running experiment which we then use to augment the existing dataset. By averaging over sets of fantasies, we can perform approximate marginalization when computing EI for a candidate point. We note that this same idea works with the constraint network, where instead of computing marginalized EI, we would compute the marginalized probability of violating a constraint.

To that end, given currently running jobs with inputs $\{\mathbf{x}_j\}_{j=1}^J$, the marginalized acquisition function $a_{\text{MCEI}}(\cdot; \mathcal{D}, \boldsymbol{\Theta}, \Psi)$ is given by

$$\begin{aligned} &a_{\text{MCEI}}(\mathbf{x}; \mathcal{D}, \{\mathbf{x}_j\}_{j=1}^J, \boldsymbol{\Theta}, \Psi) = \\ &\quad \int a_{\text{CEI}}(\mathbf{x}; \mathcal{D} \cup \{(\mathbf{x}_j, y_j)\}_{j=1}^J, \boldsymbol{\Theta}, \Psi) \\ &\quad \times \mathbb{P}\left[\{c_j, y_j\}_{j=1}^J \,|\, \mathcal{D}, \{\mathbf{x}\}_{j=1}^J\right] \mathrm{d}y_1 \ldots \mathrm{d}y_n \mathrm{d}c_1 \ldots \mathrm{d}c_n \, . \end{aligned}$$

When this strategy is applied to a GP, the cost of computing EI for a candidate point becomes cubic in the size of the augmented dataset. This restricts both the number of running experiments that can be tolerated, as well as the number of fantasy sets used for marginalization. With DNGO it is possible to scale both of these up to accommodate a much higher degree of parallelism.

Finally, following the approach of Snoek et al. (2012) we integrate out the hyperparameters of the model to obtain our final integrated acquisition function. For each iteration of the optimization routine we pick the next input, $\mathbf{x}^*$, to evaluate according to

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} a_{\text{MCEI}}(\mathbf{x}; \mathcal{D}, \{\mathbf{x}_j\}_{j=1}^J) \, , \tag{6.11}$$

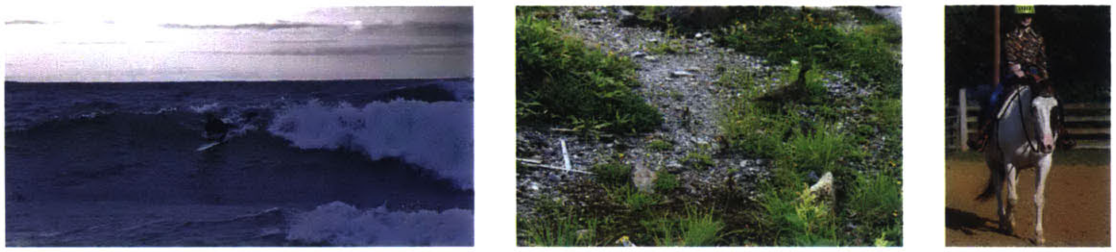

(a) "A person riding a wave in the ocean." (b) "A bird sitting on top of a field." (c) "A horse is riding a horse."

Figure 6-3: Sample test images and generated captions from the best LBL model on the COCO 2014 dataset. The first two captions sensibly describe the contents of their respective images, while the third is offensively inaccurate.

where

$$a_{\mathrm{MCEI}}(\mathbf{x};\mathcal{D},\{\mathbf{x}_j\}_{j=1}^{J}) = \int a_{\mathrm{MCEI}}(\mathbf{x};\mathcal{D},\{\mathbf{x}_j\}_{j=1}^{J},\boldsymbol{\Theta},\boldsymbol{\Psi})\,\mathrm{d}\boldsymbol{\Theta}\mathrm{d}\boldsymbol{\Psi}. \tag{6.12}$$

6.4 Experiments

6.4.1 HPOLib Benchmarks

In the literature, there exist several other methods for model-based optimization. Among these, the most popular variants in machine learning are the random forest-based SMAC procedure Hutter et al. (2011) and the tree Parzen estimator (TPE) Bergstra et al. (2011). These are faster to fit than a Gaussian process and scale more gracefully with large datasets, but this comes at the cost of a more heuristic treatment of uncertainty. By contrast, DNGO provides a balance between scalability and the Bayesian marginalization of model parameters and hyperparameters.

To demonstrate the effectiveness of our approach, we compare DNGO to these scalable model-based optimization variants, as well as the input-warped Gaussian process method of Snoek et al. (2014) on the benchmark set of continuous problems from the HPOLib package Eggensperger et al. (2013). As Table 6.1 shows, DNGO

Method	Test BLEU
Human Expert LBL	24.3
Regularized LSTM	24.3
Soft-Attention LSTM	24.3
10 LSTM ensemble	24.4
Hard-Attention LSTM	25.0
Single LBL	**25.1**
2 LBL ensemble	**25.9**
3 LBL ensemble	**26.7**

Table 6.2: Image caption generation results using BLEU-4 on the Microsoft COCO 2014 test set. Regularized and ensembled LSTM results are reported in **Zaremba et al. (2015)**. The baseline LBL tuned by a human expert and the Soft and Hard Attention models are reported in **Xu et al. (2015)**. We see that ensembling our top models resulting from the optimization further improves results significantly. We noticed that there were distinct multiple local optima in the hyperparameter space, which may explain the dramatic improvement from ensembling a small number of models.

significantly outperforms SMAC and TPE, and is competitive with the Gaussian process approach. This shows that, despite vast improvements in scalability, DNGO retains the statistical efficiency of the Gaussian process method in terms of the number of evaluations required to find the minimum.

6.4.2 Image Caption Generation

In this experiment, we explore the effectiveness of DNGO on a practical and expensive problem where highly parallel evaluation is necessary to make progress in a reasonable amount of time. We consider the task of image caption generation using multi-modal neural language models. Specifically, we optimize the hyperparameters of the log-bilinear model (LBL) from **Kiros et al. (2014)** to optimize the BLEU score of a validation set from the recently released COCO dataset **Lin et al. (2014)**. From our experiments, each evaluation of this model took an average of 26.6 hours.

We optimize learning parameters such as learning rate, momentum and batch size; regularization parameters like dropout and weight decay for word and image representations; and architectural parameters such as the context size, whether to use the additive or multiplicative version, the size of the word embeddings and the multi-modal representation size [1]. The final parameter is the number of factors, which is only relevant for the multiplicative model. This adds an interesting challenge, since it is only relevant for half of the hyperparameter space. This gives a total of 11 hyperparameters. Even though this number seems small, this problem offers a number of challenges which render its optimization quite difficult. For example, in order to not lose any generality, we choose broad box constraints for the hyperparameters; this, however, renders most of the volume of the model space infeasible. In addition, quite a few of the hyperparameters are categorical, which introduces severe non-stationarities in the objective surface.

Nevertheless, one of the advantages of a scalable method is the ability to highly parallelize hyperparameter optimization. In this way, high quality settings can be found after only a few sequential steps. To test DNGO in this scenario, we optimize the log-bilinear model with up to 800 parallel evaluations.

Running between 300 and 800 experiments in parallel (determined by cluster availability), we proposed and evaluated approximately 2500 experiments—the equivalent of over 2700 CPU days—in less than one week. Using the BLEU-4 metric, we optimized the validation set performance and the best LBL model found by DNGO outperforms recently proposed models using LSTM recurrent neural networks Zaremba et al. (2015); Xu et al. (2015) on the test set. This is remarkable, as the LBL is a relatively simple approach. Ensembling this top model with the second and third best (under the validation metric) LBL models resulted in a test-set BLEU score [2] of 26.7, significantly outperforming the LSTM-based approaches. We noticed that there were distinct multiple local optima in the hyperparameter space, which may explain the dramatic improvement from ensembling a small number of models. We show qualitative examples of generated captions on test images in Figure 6-3.

[1]Details are provided in the supplementary material.

[2]We have verified that our BLEU score evaluation is consistent across reported results. We used a beam search decoding for our test predictions with the LBL model.

Method	CIFAR-10	CIFAR-100
Maxout	9.38%	38.57%
DropConnect	9.32%	N/A
Network in network	8.81%	35.68%
Deeply supervised	7.97%	34.57%
ALL-CNN	7.25%	33.71%
Tuned CNN	**6.37%**	**27.4%**

Table 6.3: We use our algorithm to optimize validation set error as a function of various hyperparameters of a convolutional neural network. We report the test errors of the models with the optimal hyperparameter configurations, as compared to current state-of-the-art results.

6.4.3 Deep Convolutional Neural Networks

Finally, we use DNGO on a pair of highly competitive deep learning visual object recognition benchmark problems. We tune the hyperparameters of a deep convolutional neural network on the CIFAR-10 and CIFAR-100 datasets. Our approach is to establish a single, generic architecture, and specialize it to various tasks via individualized hyperparameter tuning. As such, for both datasets, we employed the same generic architecture inspired by the configuration proposed in Springenberg et al. (2014), which was shown to attain strong classification results. This architecture is detailed in the supplementary.

For this architecture, we tuned the momentum, learning rate, ℓ_2 weight decay coefficients, dropout rates, standard deviations of the random i.i.d. Gaussian weight initializations, and corruption bounds for various data augmentations: global perturbations of hue, saturation and value, random scalings, input pixel dropout and random horizontal reflections. We optimized these over a validation set of 10,000 examples drawn from the training set, running each network for 200 epochs. See Figure 6-4 for a visualization of the hyperparameter tuning procedure.

We performed the optimization on a cluster of Intel® Xeon Phi™ coprocessors, with 40 jobs running in parallel using a kernel library that has been highly optimized for efficient computation on the Intel® Xeon Phi™ coprocessor[3]. For the optimal

[3]Available at `https://github.com/orippel/micmat`

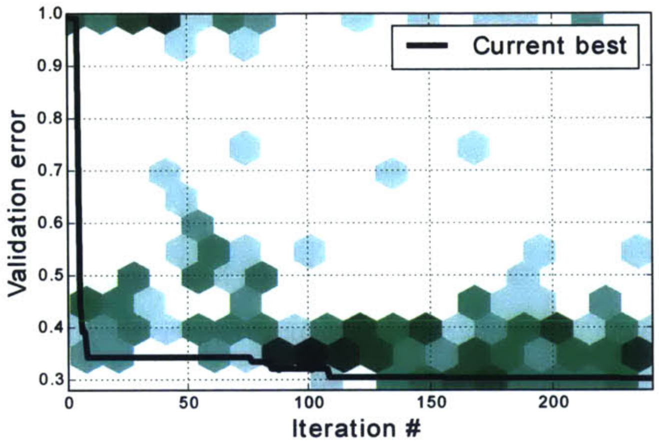


Figure 6-4: Validation errors on CIFAR-100 corresponding to different hyperparameter configurations as evaluated over time. These are represented as a planar histogram, where the shade of each bin indicates the total count within it. The current best validation error discovered is traced in black. This projection demonstrates the exploration-versus-exploitation paradigm of Bayesian Optimization, in which the algorithm trades off visiting unexplored parts of the space, and focusing on parts which show promise.

hyperparameter configuration found, we ran a final experiment for 350 epochs on the entire training set, and report its result.

Our optimal models for CIFAR-10 and CIFAR-100 achieved test errors of 6.37% and 27.4% respectively. A comparison to published state-of-the-art results Goodfellow et al. (2013); Wan et al. (2013); Lin et al. (2013); Lee et al. (2014); Springenberg et al. (2014) can be found in Table 6.3.

A comprehensive overview of the setup, the architecture, the tuning and the optimum configuration can be found in the supplementary material.

6.5 Discussion and Remaining Open Problems

In this paper, we introduced deep networks for global optimization, or DNGO, which enables efficient optimization of noisy, expensive black-box functions. While this model maintains desirable properties of the GP such as tractability and principled management of uncertainty, it greatly improves its scalability from cubic to linear as a function of the number of observations. We demonstrate that while this model allows efficient computation, its performance is nevertheless competitive with existing state-of-the-art approaches for Bayesian optimization. We demonstrate empirically that it is especially well suited to massively parallel hyperparameter optimization.

While adaptive basis regression with neural networks provides one approach to the enhancement of scalability, other models may also present promise. One promising line of work, for example by Nickson et al. (2014), is to introduce a similar methodology by instead employing the sparse Gaussian process as the underlying probabilistic model Snelson & Ghahramani (2005); Titsias (2009); Hensman et al. (2013).

Chapter 7

High-Dimensional Probability Estimation with Deep Density Models

7.1 Introduction

Many core machine learning tasks are concerned with density estimation and manifold discovery. Probabilistic graphical models are a dominating approach for constructing sophisticated density estimates, but they often present computational difficulties in practice. For example, undirected models, such as the Boltzmann machine (Smolensky, 1986; Hinton et al., 2006) are able to achieve compact and efficiently-computed latent variable representations at the cost of only providing unnormalized density estimates. Directed belief networks (Pearl, 1988; Neal, 1992; Adams et al., 2010), on the other hand, enable one to specify *a priori* marginals of hidden variables and are easily normalized, but require costly inference procedures. Bayesian nonparametric density estimation (e.g., Escobar & West (1995); Rasmussen (2000); Adams et al. (2009)) is another flexible approach, but it often requires costly inference procedures and does not typically scale well to high-dimensional data.

Manifold learning provides an alternative way to implicitly characterize the density of data via a low-dimensional embedding, e.g., locally-linear embeddding (Roweis & Saul, 2000), IsoMap (Tenenbaum et al., 2000), the Gaussian process latent variable model (Lawrence, 2005), kernel PCA (Schölkopf et al., 1998), and t-SNE (van der

Maaten & Hinton, 2008). Typically, however, these methods have emphasized visualization as the primary motivation. A notable exception is the autoencoder neural network (Cottrell et al., 1987; Hinton & Salakhutdinov, 2006), which seeks embeddings in representation spaces that themselves can be high dimensional. Unfortunately, the autoencoder does not have a clear probabilistic interpretation (although see Rifai et al. (2012) for a discussion).

Some approaches, such as manifold Parzen windows (Vincent & Bengio, 2002), have attempted to tackle the combined problem of density estimation and manifold learning directly, but have faced difficulties due to the curse of dimensionality. Other approaches, such as the Bayesian GP-LVM (Titsias & Lawrence, 2010), characterize the manifold implicitly in terms of a nonlinear mapping from a representation space to the observed space. To define a coherent probabilistic model, however, it is necessary to find an invertible map between these spaces so that the density of a datum can be evaluated in the latent space without integrating over the pre-image. It has proven difficult to flexibly parameterize the space of such invertible maps, however, let alone find a transformation that results in a tractable density on the representation space. Independent components analysis (Bell & Sejnowski, 1995) and the related idea of a density network (MacKay & Gibbs, 1997) are examples of bijective models that exploit invertible linear transformations; these, however, have rather limited expressiveness. DiffeoMap (Walder & Schölkopf, 2008) establishes a bijection close to a lower-dimensional subspace, and then projects to it. Other approaches, such as the the back-constrained GP-LVM (Lawrence & Candela, 2006) attempt to approximate this bijection.

In this work, we introduce the *deep density model* (DDM), an approach that bridges manifold discovery and density estimation. We exploit ideas from deep learning to introduce a rich and flexible class of bijective transformations of the observed space. We optimize over these transformations to obtain a map under which the implied distribution on the representation space has an approximately factorized form with known marginals. The invertibility of the map ensures that measure is not collapsed across the transformation, and as such, the determinant of the Jacobian can be computed. This leads to fully-normalized probability densities without a partition function.

The combination of rich bijective transformations with density estimation enables us to explore a variety of modeling directions for high-dimensional data. As the approach is generative, we can easily sample data from a trained model without Markov chain Monte Carlo (MCMC). We present a variety of applications to the CIFAR and MNIST datasets, for proof-of-concept. The deep density model also provides new possibilities for supervised learning by building Bayesian classifiers that have well-calibrated class-conditional probabilities. This additionally permits to exploit densities of unlabeled data to perform unsupervised learning, by constructing mixtures of models and training them coherently with expectation maximization.

In developing the deep density model, we also provide insight into a variety of fundamental concepts for latent variable models. Using information theoretic tools, we identify important connections between sparsity and the independence of the latent dimensions. These connections allow finding a map leads to an approximately factorized latent distribution. By understanding the distribution of the data in representation space and the transformation that gives rise to it, we can characterize the entropy of the distribution over data in the observed space. This enables making informed choices in model selection.

7.2 Bijections and Normalized Densities

We are interested in learning a distribution over data in a high-dimensional space $\mathscr{Y} \subseteq \mathbb{R}^K$. We denote this (unknown) distribution as $p_{\boldsymbol{Y}}(\cdot)$. An axiomatic assumption in machine learning is that the data contain structure, and this corresponds to $p_{\boldsymbol{Y}}(\cdot)$ distribution having most of its probability mass on a lower-dimensional, but very complicated, manifold in $\mathscr{Y}$. Tractably parametrizing the space of such manifolds and then fitting the resulting distributions to data is a significant challenge.

Similarly, studying this distribution directly in the observed space presents both theoretical and computational difficulties. Instead, we consider how the data might be the result of a transformation from an unobserved *representation space* $\mathscr{X} \subseteq \mathbb{R}^K$. We denote the distribution on this space as $p_{\mathbf{X}}(\cdot)$, and we assume the observed data arise from a transformation $\boldsymbol{f} : \mathscr{X} \to \mathscr{Y}$. We assume that the latent distribution $p_{\mathbf{X}}(\cdot)$ has a simple factorized form:

$$p_{\mathbf{X}}(\mathbf{x}) = \prod_{k=1}^{K} p_{X_k}(x_k) \,, \tag{7.1}$$

where the marginal factors $p_{X_k}(\cdot)$ have a simple and known univariate form.

We further make the assumption that $\boldsymbol{f}(\cdot)$ is bijective and that $\boldsymbol{f}^{-1}(\cdot)$ is available analytically. In this case, the probability density for a point $\boldsymbol{y} \in \mathscr{Y}$ can be computed:

$$p_{\mathbf{Y}}(\mathbf{y}) = \prod_{k=1}^{K} p_{X_k}([\boldsymbol{f}^{-1}(\mathbf{y})]_k) \left| \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \mathbf{x}} \right|_{\boldsymbol{f}^{-1}(\mathbf{y})} \,. \tag{7.2}$$

In this paper, we introduce the *deep density model*, which discovers rich bijective transformations to map from simple latent distributions into complex observed densities. By optimizing over a large and flexible class of such bijective transformations, it is possible to discover structure in high-dimensional data sets while still having a manageable, normalized density estimator.

Bijectivity is critical for ensuring that the density in Eq. (7.2) is normalized. This bijectivity is in contrast to many neural network approaches to latent representation where the latent space is often smaller (for an information bottleneck in, e.g., an autoencoder) or larger (for an *overcomplete* representation) than the observed space. When the representation space is smaller, then $\boldsymbol{f}(\cdot)$ cannot be surjective and so multiple points in $\mathscr{Y}$ may map to the same point in $\mathscr{X}$, leading to overestimates of the density. In the overcomplete case, we also sacrifice bijectivity since $\boldsymbol{f}^{-1}(\cdot)$ cannot be surjective; in other words, the image of $\mathscr{Y}$ under $\boldsymbol{f}^{-1}(\cdot)$ will not span $\mathscr{X}$. As such, $p_{\mathbf{X}}(\cdot)$ will have support beyond $\boldsymbol{f}^{-1}(\mathscr{Y})$. That is, the latent normalization includes mass that appears in $\mathscr{X} \setminus \{\boldsymbol{f}^{-1}(\mathscr{Y})\}$. However, taking this mass into account is a very challenging problem, whose difficulty unfortunately increases with the richness

7.3.2 Divergence Penalty: Sculpting the Latent Marginals

In order to fit the deep density model to data, it is necessary to specify a measure of distance between the empirical distribution and the model distribution. We achieve this via a divergence penalty, which forces the model to distribute the mass of the data in the representation space so as to be similar to a distribution chosen *a priori*. This construction has several advantages: it 1) results in a known, fixed distribution on the representation space that can be used to generate fantasy data, 2) enables sparsity to be enforced as a constraint rather than a penalty weighed against the reconstruction cost, and 3) combats overfitting by explicitly requiring that some of the data have low probability under the model. This third advantage is subtle, but critical: some data must live in the tail of the distribution, in contrast to the maximum of the posterior which is simply a weighing of the MLE against the mode of the prior. See Figure 7-1 for a comparison of distributions produced by a various regularization techniques.

Concretely, we assume that the representation space is a unit hypercube, i.e., $\mathscr{X} = [0,1]^K$. As before, we assume there are N data $\{\mathbf{y}_n\}_{n=1}^N$, which (for a given $\mathbf{\Psi}$) are mapped into $\mathscr{X}$ to give $\{\mathbf{x}_n = \boldsymbol{g}_{\mathbf{\Psi}}(\mathbf{y}_n)\}_{n=1}^N$. Ideally, for representation dimension $k \in 1, \dots, K$, the divergence penalty would measure the difference between the marginal empirical distribution

$$\hat{p}_{X_k}(x) = \frac{1}{N} \sum_{n=1}^{N} \delta(x - [\mathbf{x}_n]_k) \tag{7.7}$$

and a target univariate distribution $q(\cdot)$ that we define. In practice, we approximate $\hat{p}_{X_k}(\cdot)$ by finding the best fit of a tractable parametric family and then computing the symmetrized Kullback–Liebler divergence:

$$T\,(p(\cdot)\,||\,q(\cdot)) = D(p(\cdot)\,||\,q(\cdot)) + D(q(\cdot)\,||\,p(\cdot))\,, \tag{7.8}$$

maps the observed data into the representation space. This function will be optimized to imbue the latent distribution $p_{\boldsymbol{X}}(\cdot)$ with the properties outlined in Section 7.2; the necessity of $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ will be expanded upon in Subsection 7.3.4. $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ is composed of J layers, and the j-th layer of $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ has K_j hidden units (with $K_J = K$) and parameters $\boldsymbol{\Gamma}_j$ and $\boldsymbol{\gamma}_j$. We denote the parameters for $\boldsymbol{g}$ as $\boldsymbol{\Psi} := \bigcup_{j=1}^{J}\{\boldsymbol{\Gamma}_j, \boldsymbol{\gamma}_j\}$ and use the notational shortcuts above to write

$$\boldsymbol{g}_{\boldsymbol{\Psi}}(\boldsymbol{y}) = \bigcirc_{j=1}^{J} \boldsymbol{S}_{\boldsymbol{\Gamma}_j, \boldsymbol{\gamma}_j}(\boldsymbol{y})\,. \tag{7.5}$$

7.3.1 Regularizing the Transformations

We will train the model on data by minimizing an objective composed of several parts:

Divergence Penalty $\mathcal{D}(\boldsymbol{\Psi})$: This determines the fit of the current encoding transformation. It forces the marginal densities of the empirical distribution of the representation-space data to match a target distribution of our choice, by penalizing divergence from it.

Invertibility Measure $\mathcal{I}(\boldsymbol{\Theta})$: This ensures the invertibility of $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$ by penalizing poorly-conditioned transformations.

Reconstruction Loss $\mathcal{R}(\boldsymbol{\Theta}, \boldsymbol{\Psi})$: This jointly penalizes the encoder $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ and decoder $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$ to ensure that $\boldsymbol{g}_{\boldsymbol{\Psi}}(\boldsymbol{y}) \approx \boldsymbol{f}_{\boldsymbol{\Theta}}^{-1}(\boldsymbol{y})$ on the data.

Each of these participates in the overall objective given by:

$$C(\boldsymbol{\Theta}, \boldsymbol{\Psi}) = \mu_{\mathcal{D}}\mathcal{D}(\boldsymbol{\Theta}) + \mu_{\mathcal{I}}\mathcal{I}(\boldsymbol{\Psi}) + \mu_{\mathcal{R}}\mathcal{R}(\boldsymbol{\Theta}, \boldsymbol{\Psi})\,, \tag{7.6}$$

where $\mu_{\mathcal{I}}, \mu_{\mathcal{D}}, \mu_{\mathcal{R}} \in \mathbb{R}$ are the weights of each term. We will examine each of these terms in more detail in the proceeding sections.

7.3.2 Divergence Penalty: Sculpting the Latent Marginals

In order to fit the deep density model to data, it is necessary to specify a measure of distance between the empirical distribution and the model distribution. We achieve this via a divergence penalty, which forces the model to distribute the mass of the data in the representation space so as to be similar to a distribution chosen *a priori*. This construction has several advantages: it 1) results in a known, fixed distribution on the representation space that can be used to generate fantasy data, 2) enables sparsity to be enforced as a constraint rather than a penalty weighed against the reconstruction cost, and 3) combats overfitting by explicitly requiring that some of the data have low probability under the model. This third advantage is subtle, but critical: some data must live in the tail of the distribution, in contrast to the maximum of the posterior which is simply a weighing of the MLE against the mode of the prior. See Figure 7-1 for a comparison of distributions produced by a various regularization techniques.

Concretely, we assume that the representation space is a unit hypercube, i.e., $\mathscr{X} = [0,1]^K$. As before, we assume there are N data $\{\mathbf{y}_n\}_{n=1}^N$, which (for a given $\mathbf{\Psi}$) are mapped into $\mathscr{X}$ to give $\{\mathbf{x}_n = \boldsymbol{g}_{\mathbf{\Psi}}(\mathbf{y}_n)\}_{n=1}^N$. Ideally, for representation dimension $k \in 1, \ldots, K$, the divergence penalty would measure the difference between the marginal empirical distribution

$$\hat{p}_{X_k}(x) = \frac{1}{N}\sum_{n=1}^{N} \delta(x - [\mathbf{x}_n]_k) \tag{7.7}$$

and a target univariate distribution $q(\cdot)$ that we define. In practice, we approximate $\hat{p}_{X_k}(\cdot)$ by finding the best fit of a tractable parametric family and then computing the symmetrized Kullback–Liebler divergence:

$$T\left(p(\cdot)\,||\,q(\cdot)\right) = D(p(\cdot)\,||\,q(\cdot)) + D(q(\cdot)\,||\,p(\cdot))\,, \tag{7.8}$$

where

$$D(p(\cdot)\,||\,q(\cdot)) = \int_{\mathscr{X}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}\, \mathrm{d}\mathbf{x}. \tag{7.9}$$

Since, in this case the representation space is the unit hypercube, we choose our objective distribution to be a member of the Beta family:

$$q(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}. \tag{7.10}$$

Given the data representations $\{\mathbf{x}_n\}_{n=1}^N$, we estimate the empirical distribution of each dimension with a Beta distribution, using moment-matching to approximate its parameters:

$$\hat{\alpha}_k = \hat{\mu}_k \left[\frac{\hat{\mu}_k\,(1-\hat{\mu}_k)}{\hat{\sigma}_k^2} - 1\right] \tag{7.11}$$

$$\hat{\beta}_k = (1-\hat{\mu}_k) \left[\frac{\hat{\mu}_k\,(1-\hat{\mu}_k)}{\hat{\sigma}_k^2} - 1\right], \tag{7.12}$$

where $\hat{\mu}_k$ and $\hat{\sigma}_k^2$ are the sample mean and variance, respectively. We note that a Beta distribution with parameter $\alpha < 1$ produces a very sharp peak at 0, and as such allows to pursue sparsity in distribution (under the assumption that elements of small magnitude cannot be distinguished from each other).

With these in hand, we get a closed-form expression for our divergence penalty:

$$\begin{aligned} T\left(\hat{p}_{X_k}(\cdot)\,||\,q(\cdot)\right) &= (\hat{\alpha}-\alpha)\left[\psi(\hat{\alpha})-\psi(\alpha)\right] + \left(\hat{\beta}-\beta\right)\left[\psi(\hat{\beta})-\psi(\beta)\right] && (7.13)\\ &\quad - \left(\hat{\alpha}-\alpha+\hat{\beta}-\beta\right)\left[\psi(\hat{\alpha}+\hat{\beta})-\psi(\alpha+\beta)\right], && (7.14)\end{aligned}$$

where $\psi(z) = \mathrm{d}\log\Gamma(z)/\mathrm{d}z$ is the digamma function.

Furthermore, for each example, we impose an example divergence penalty, denoted as $\hat{p}_{X_n}(\cdot)$, which penalizes the distance between our objective distribution and the empirical distribution over the elements of that particular example:

Finally, our total divergence penalty is

$$\mathcal{D}(\Psi) = \frac{1}{K}\sum_{k=1}^{K} T\left(\hat{p}_{X_k}(\cdot)\,||\,q(\cdot)\right) + \frac{1}{N}\sum_{n=1}^{N} T\left(\hat{p}_{X_n}(\cdot)\,||\,q(\cdot)\right) .$$

Sparsity in Distribution

The traditional pursuit of sparsity entails the application of an L_1-type regularization that directly penalizes the activations in the representation space. This has several undesirable properties. First, the penalty does not differentiate between the cases where the activated units are distributed evenly among examples, and where a fixed set of units is always activated at all examples while others never are. Furthermore, it is discomforting that, in the limit of small reconstruction cost in the objective function, the regularization term is optimized if and only if all examples are identically mapped to the same point, namely zero. This forces *all* the activations to be small in order to have some of them vanish; it artificially forces the activation distribution to be contained in a small region around zero. Another implication of direct activity penalization is that we must search the parameter space of the regularization coefficient in order to attain our desired sparsity structure.

Instead of inducing sparsity directly, we achieve it in *distribution*, across examples. In practice, this arises from penalizing the KL-divergence between the *empirical distribution* — the actual distribution in the representation space for the given set of observations — and an appropriately-chosen target distribution $q(\cdot)$, which has a peak at 0. The difference between this and the traditional L_1 approach to sparsity can be understood by considering the optimization problem as the dualization of the sparsity constraints. In the case of an L_1-type regularization, these constraints directly bound the space under the prescribed distance metric. In the L_1 case, we thus have a situation with two different points on the same contour of these constraints, one of which a more desirable of a solution than the other. On the other hand, the constrains that emerge from the divergence penalization are imposed within the probability simplex. The advantage is that a contour of these constraints corresponds to a locus of distributions with similar sparsity structures and thus similar desirability.

7.3.3 Invertibility: condition number penalty

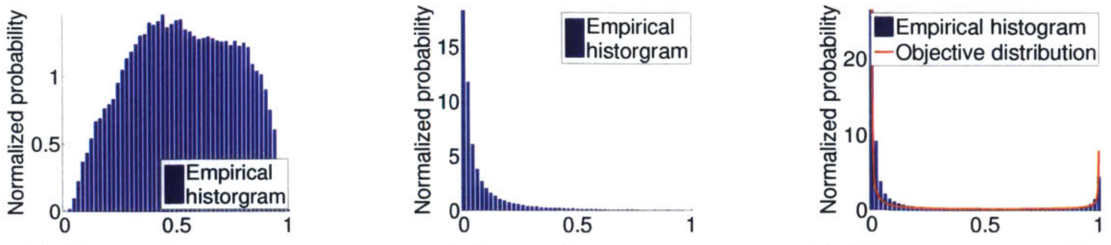
Invertibility of $\boldsymbol{f}_{\boldsymbol{\Theta}}$ is critical to providing a normalized density. A standard autoencoder contracts volumes around observed examples only, due to the reconstruction penalty approximating an invertible map at the observations. A true bijection, however, will guarantee conservation of volume not just at the data, but also at points we have never seen before. This will allow computation of the determinant of the Jacobian of $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$, and precisely specify how probability mass is reshuffled by the transformation. To ensure invertibility of the transformation, we must ensure the invertibility of each layer. As the nonlinear activation functions are fixed, invertibility is determined by the condition numbers of the $\boldsymbol{\Omega}_m$ matrices in $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$. We therefore introduce a regularization term that ensures invertibility:

$$\mathcal{I}(\Theta) = \frac{1}{M} \sum_{m=1}^{M} \log \left(\frac{\lambda_{\max}(\boldsymbol{\Omega}_m)}{\lambda_{\min}(\boldsymbol{\Omega}_m)} \right). \tag{7.15}$$

Here, $\lambda_{\max}(\boldsymbol{A})$ and $\lambda_{\min}(\boldsymbol{A})$ are the maximum and minimum eigenvalues of $\boldsymbol{A}$. In this case, the curse of dimensionality becomes a blessing of dimensionality: in high dimensions, not only orthogonality is easily attained, it is difficult to escape. We find that in practice the invertibility requirement is easily satisfied, and does not constrain the algorithm at all.

7.3.4 Reconstruction and the independence of latent dimensions

Since we now have the ability to dictate the marginal distributions in the representation space, we can shed light on the connection between entropy, sparsity and independence as a function of the transformation to the representation space. In order to attain a tractable distribution in the representation space, we must eliminate dependencies between the latent dimensions. We refer to this process as *diversifica-*



(a) No regularization (b) L_1 regularization (c) Divergence penalization

Figure 7-1: Histograms of the empirical distribution at $\hat{p}_{X_1}(\cdot)$, $k = 1$ with the specified methods of regularization, upon training on MNIST. The objective distribution was taken as Beta $(\cdot; 0.02, 0.2)$.

tion, as it reduces the overlap of information learnt by distinct dimensions; see the effects of this process in Figure 7-2.

Diversification arises naturally when considering the effect of simultaneously increasing sparsity in the latent representation, while decreasing the entropy of the target marginals. The reconstruction penalty demands that information be preserved in the representation space. However, as the entropy decreases, the information capacity of the marginals decreases; it is necessary for the dimensions of the latent distribution to become more independent in order to reduce redundancy and continue to reconstruct successfully. Hence, we increase sparsity by limiting the capacity of the encoder, until we get to the point of minimum marginal entropy under sufficient conservation of information (which we measure by reconstruction of the observations). At this point, we expect the marginals to be approximately independent.

In the second line below, notice that we may write the observed space entropy in terms of the representation entropy and a term that accounts for the contraction of volumes under the transformation. We then write the joint entropy in terms of the marginal entropies and the mutual information information between the joint distribution and the independent marginal factorization:

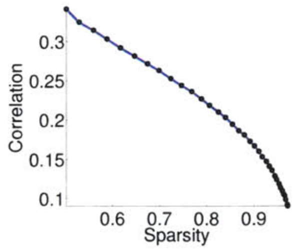
$$\begin{aligned}
\mathcal{H}(p_{\boldsymbol{Y}}(\cdot)) &= -\int_{\mathcal{Y}} p_{\boldsymbol{Y}}(\mathbf{y}) \log p_{\boldsymbol{Y}}(\mathbf{y}) d\mathbf{y} \\
&= \mathcal{H}(p_{\boldsymbol{X}}(\cdot)) + \mathbb{E}_{\mathscr{X}}\left[\log\left|\frac{\partial \boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)}{\partial \mathbf{y}}\right|\right] \\
&= \sum_{k=1}^{K} \mathcal{H}(p_{X_k}(\cdot)) - D(\prod_{k=1}^{K} p_{X_k}(\cdot) \,||\, p_{\boldsymbol{X}}(\cdot)) + \mathbb{E}_{\mathscr{X}}\left[\log\left|\frac{\partial \boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)}{\partial \mathbf{y}}\right|\right] \qquad (7.16)
\end{aligned}$$

The latent dimensions are independent if and only if the mutual information is zero, and as such, we seek to minimize it. The entropy in the observed space is fixed, but we have control of the marginal entropies, and we can decrease $\mathbb{E}_{\mathscr{X}}\left[\log\left|\frac{\partial \boldsymbol{f}(\cdot)}{\partial \mathbf{y}}\right|\right]$ by placing an information bottleneck on $\boldsymbol{f}(\cdot)$. Thus, by minimizing both these terms, we can minimize latent dependencies.

To that end, we approximate

$$\mathbb{E}_{\mathscr{X}}\left[\log\left|\frac{\partial \boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)}{\partial \mathbf{y}}\right|\right] \approx \frac{1}{N}\sum_{n=1}^{N} \log\left|\frac{\partial \boldsymbol{f}_{\boldsymbol{\Theta}}(\mathbf{x}_n)}{\partial \mathbf{y}}\right| . \qquad (7.17)$$

Note that an inherent property of the sigmoid nonlinearity is its asymptotic flatness as its output approaches zero. As such, the decoder $\boldsymbol{f}_{\boldsymbol{\Theta}}$ becomes more unable to distinguish between points as their magnitude decreases. Thus, for each representation dimension, by increasing the α parameter of our objective distribution $q(\cdot;\alpha,\beta)$, we can shift more probability mass towards zero: this not only decreases the marginal entropies $\sum_{k=1}^{K} \mathcal{H}(p_{X_k}(\cdot))$, but also increases the information bottleneck. We add that the flatness of the sigmoid still does not imply true sparsity: a deep transformation can distinguish between small but nonzero values, and as such "undo" the sparsity effect—information is not gone, only bit-shifted. As such, in order to ensure true information loss in the neighbourhood of zero, we also introduce a threshold to the encoder, that kills any representation element less than some $\varepsilon > 0$; namely, we use the encoder $\mathbb{I}_{\boldsymbol{g}_{\Psi}(\mathbf{y})\geq\varepsilon}\boldsymbol{g}_{\Psi}(\mathbf{y})$.



(a)

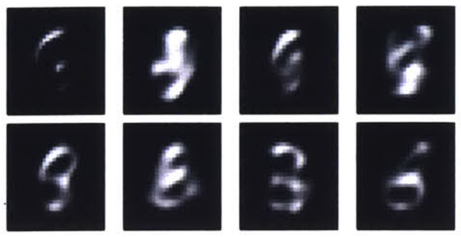

(b)

Figure 7-2: Effects of diversification. (a) Decrease of first-order dependencies as function of sparsity. (b) Generations from the model by sampling independently from the latent marginals. First row: generations from a model with high-entropy marginals $\text{Beta}(\cdot; 0.02, 0.2)$ with strong dependence; images are incoherent. Second row: generations from a model with diversified marginals with a final distribution $\text{Beta}(\cdot; 0.004, 0.2)$.

Entropy characterization

A direct consequence of the above is our ability to now place an upper bound on $\mathcal{H}(p_{\boldsymbol{Y}}(\cdot))$, and, once the mutual information is minimized, to characterize it. This now allows us to make informed choice of a representation space—for example, we understand the interplay between the latent dimensionality and sparsity. Minimizing the mutual information in the way demonstrated above corresponds to selecting this space to be maximally sparse under the constraint of retaining the information encapsulated in the input examples.

We furthermore note that the entropy of model can be thought of as the "effective number of configurations" that data drawn from its distribution can take. As such, it governs how mass is allocated between the training examples and out-of-sample data: thus, the diversification procedure described dictates the generalizability of the model in a very transparent way.

On why we need $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$

We can now understand the motivation for introducing $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ even with a bijective map. We need $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ to induce points in the observed space to be close together in $\mathscr{X}$, both to control the information capacity as well as to determine the marginals in $\mathscr{X}$.

On the other hand, $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$ should not expand measure in mapping from $\mathscr{X}$ back to $\mathscr{Y}$; this ensures that there is an information bottleneck even under bijectivity. However, asking $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$ and $\boldsymbol{f}_{\boldsymbol{\Theta}}^{-1}(\cdot)$ to serve this double purpose is contradictory, as one function exactly undoes the contraction of measure performed by the other function. Furthermore, $\boldsymbol{g}_{\boldsymbol{\Psi}}(\cdot)$ is helpful from a computational point of view. In the process of shaping the latent distribution, we must define our requirements as penalties on $\mathscr{X}$, which we proceed to back-propagate. Since asymptotic flatness of $\boldsymbol{f}_{\boldsymbol{\Theta}}(\cdot)$ means asymptotic steepness of $\boldsymbol{f}_{\boldsymbol{\Theta}}^{-1}(\cdot)$, performing numerical optimization on this function—let alone demanding representation points to be clustered in this asymptotically steep regime, as the divergence penalty requires—is clearly numerically unstable.

7.4 Training

We train the model on a GPU cluster. In cases of continuous input dimensions, we pre-process the data by whitening and normalizing it. We additionally use PCA to reduce the dimensionality of CIFAR.

In the pretraining stage, we recursively train single-layer deep density models with stochastic gradient descent, where we take the representation space examples of one iteration as the observed space examples of the next.

In the fine-tuning stage, we optimize the objective in Eq. (7.6) by performing block coordinate descent: we iterate through the layers, and for each layer we take a step to minimize the objective as a function only of that layer's parameters. Since different gradient magnitudes of distinct layers are not mixed here, this side-steps the problem of having the gradient exponentially decay during back-propagation.

The optimization procedure is designed to have few free parameters: most are actively adapted in the process of optimization. The step size is chosen adaptively via an inexact Armijo's rule line search. Exact line search on the overall objective is computationally expensive and not possible using minibatches of data. Secondly, the coefficients $\mu_{\mathcal{I}}, \mu_{\mathcal{D}}, \mu_{\mathcal{R}}$ are adapted to maintain a specified ratio of gradient magnitudes, as the step direction is a mixture of the various penalties and is thus dictated by the relative proportions of their gradients.

We sprinkle masking noise onto the inputs to attain robust solutions, as presented in Vincent et al. (2008). We also add momentum, but implicitly: we define a window size, and sweep it across the shuffled indices of the training examples, in increments that are a fraction of the window size. As such, each example will be presented in several minibatches in a row, but with each minibatch still introducing new training examples into the objective. We found this implicit momentum technique gives the algorithm a more stable convergence.

7.4.1 Distribution sequencing and initialization

Enforcing the divergence penalty immediately after initialization results in a highly nonconvex and thus a very challenging optimization problem. The diversification procedure often terminates with an objective distribution $\text{Beta}(\cdot;\alpha,\beta)$ with $\alpha \ll 1$. As such, once the algorithm settles near a solution whose empirical distribution in the representation space takes this shape, it will be very difficult for it to "reshuffle" the representation data to attain an alternative solution also with the same distribution. To that end, instead of penalizing directly with the final objective distribution, we penalize through a family of distributions $q_j(\cdot) = \text{Beta}(\cdot;\alpha_j,\beta_j)$ over iterations j, where we have $\alpha_0, \beta_0 > 1$ as an "easy" initial problem, and then have the parameter sequences converge as $\alpha_j \to \alpha, \beta_j \to \beta$. The transitions between consecutive elements in these sequences are also adaptive: the objective distribution parameters are updated once the current empirical distribution is sufficiently close to the objective distribution.

In addition, we choose the sequence of hyperparameters such that $\frac{\alpha_j}{\alpha_j+\beta_j} = \frac{\alpha}{\alpha+\beta} = \mathbb{E}[\text{Beta}(\cdot;\alpha,\beta)]$, as maintaining a constant expectation improves stability. In accordance with this, we initialize the biases as $\log\left(\frac{\alpha}{\beta}\right)$, which similarly gives rise to a consistent initial expectation. In order to initialize the weight matrices to satisfy the invertibility constraint, we select them to be random orthonormal matrices, with a scaling such that the representation distribution approximately matches in variance the initial objective distribution $\text{Beta}(\cdot;\alpha_0,\beta_0)$.

Dataset	MNIST	CIFAR-10
Training examples	3301.5	-41.8
Test examples (TE)	3343.7	-45.5
Test examples rotated by 90^0	(TE) – 63.9	(TE) – 1.9
TE with 10% of elements corrupted	(TE) – 310.6	(TE) – 123.1

Table 7.1: Mean log-probabilities of points in the observed space. The test examples are assigned similar probability as the training examples; the rotated test examples are assigned slightly less probability than the test examples; and the corrupted examples are assigned significantly lower probability. Both models were taken to have 3 layers.

7.5 Empirical Results

In this section, we examine the properties of the model and learning algorithm on benchmark data: the MNIST digits (60,000 28×28 binary handwritten numerals)[1] and the CIFAR-10 image data[2] (50,000 32×32 color images). In particular, we examine the global and local properties of the learned densities, via generation and perturbation. We emphasize that the density estimates we report here arise are fully normalized due to the tractability of the Jacobian determinant of our transformation.

7.5.1 Density evaluation

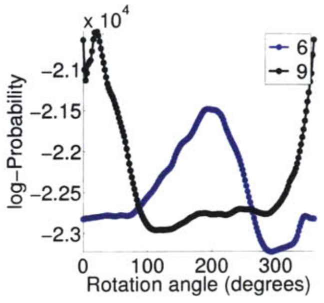
We start by conducting a variety of tests to examine the quality of the density estimates produced by the DDM.

First, for a probability model to be useful, it should not overfit by distributing most of the mass to training examples, but also assign high probability to unseen out-of-sample data. Similarly, it should assign low density to points in data space that resemble real observations, which in fact are not.

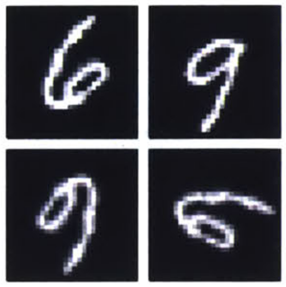
In Table 7.1, we compute the probabilities assigned by models to their training examples, test examples, and distortions of the test examples.

As an interpretable example, we train a model on examples from MNIST's digit 9

[1]`http://yann.lecun.com/exdb/mnist/`
[2]`http://www.cs.toronto.edu/~kriz/cifar.html`



(a)



(b)

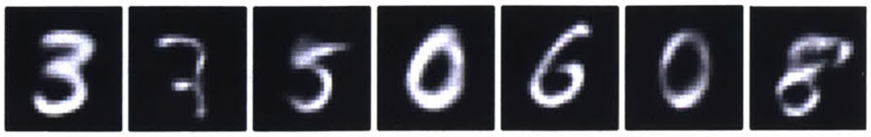
Figure 7-3: Probabilities of a rotated 6 under a model trained only on 9's. A real 9 is more probable than an inverted 6. (a) log-densities of a 6 and a 9 as a function of the angle of rotation. (b) Respectively: the original 6; a 9; the rotation of the 6 that the 9-model assigned the highest probability to; and the rotation that the 9-model assigned the lowest probability to.

class, and consider the log-probability it assigns the digit 6 under rotation. Intuitively, we expect the highest density to be assigned to the upside-down 6; see Figure 7-3. We also add a test-set 9 to demonstrate the density calibration.

We also investigate the marginal entropy of MNIST. We train the model to have a final diversification marginals $\text{Beta}\left(\cdot; 0.01, 0.4\right)$. This distribution is extremely peaked at 0 and 1, which justifies approximating the MNIST representation elements as Bernoullis by rounding off to 0 and 1 as $[x_n]_k = \left\lceil [x_n]_k - \frac{1}{2} \right\rceil$, which corresponds to Bernoulli parameter $p = \int_{1/2}^{1} \frac{\Gamma(0.01+0.4)}{\Gamma(0.01)\Gamma(0.4)} x^{0.01-1} \left(1-x\right)^{0.4-1} d\text{x} \approx 0.0224$. We now note that, by the law of large numbers, as $K \to \infty$, we expect the mean log-probability of the Bernoulli test examples to approach $-KH\left(\text{Bern}\left(\cdot; 0.0465\right)\right) = 21.0181....$ Indeed, we compute that $\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} \log \text{Bern}\left([x_n]_k; 0.0465\right) = 20.7217....$ Thus, we see that the marginal entropy of the model is close to our expectation of it.

7.5.2 Generation

The approximate independence due to the process of diversification described in Subsection 7.3.4 combined with the invertibility property of the decoder allows us to produce real samples from the density estimation on the observed space, by sampling within the representation space. Such instantiations of the distribution provide us with visualizations of regions of high density under it; see Figure 7-4.


(a) Generations from a 3-layer DDM on MNIST, with diversified marginals Beta $(\cdot; 0.015, 0.8)$.

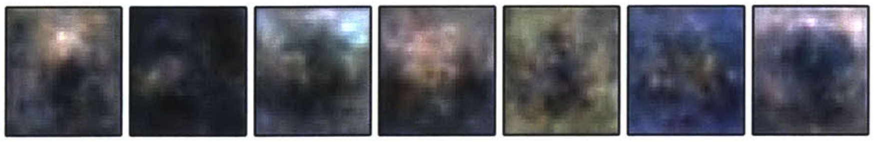

(b) Generations from a 3-layer DDM on CIFAR-10, with diversified marginals Beta $(\cdot; 0.5, 3)$.

Figure 7-4: Generations from DDMs.

7.6 Discussion and Remaining Open Problems

In this paper, we have proposed a new way to construct normalized probability density estimates from high-dimensional data. Our approach borrows ideas from deep learning, differential geometry, and information theory to ensure that the learned distributions are rich, but tractable and normalizable.

There are several interesting directions that we believe such a density model opens up. One advantage of a fully-normalized density model is that it enables Bayesian probabilistic classifiers to be constructed to provide calibrated conditional distributions over class membership. If the data are drawn from a mixture of C classes, we may compute $p(\cdot; c)$, the likelihood that an element is drawn from class c by training a model with only the examples under that class. It is then possible to, e.g., estimate the most likely class:

$$c_n^* = \arg\max_c p(\mathbf{y}_n; c) \qquad \forall n = 1, \ldots, N \,.$$

Moreover, since we now have class-conditional densities, we may reject classification of examples if no class model is confident enough to own them. Namely, we can set a threshold parameter λ, and reject classification of example $\mathbf{y}_n$ if $p(\mathbf{y}_n; c) < \lambda \; \forall c = 1, \ldots, C$.

Lastly, we can not only control the density at the observations, but also the distribution of mass in the rest of the latent space. Specifically, instead of letting each class model see only its own examples, we can expose it to training examples from other classes, and demand it assigns them as little probability as possible. This not only teaches the model to recognize examples that lie on the manifold of its class, but also identify differences from other examples by mapping them away from this manifold.

We tested the above ideas on MNIST, and found that a raw mixture of Bayesian classifiers gives us an error rate of 9.5%. However, penalizing density assigned to foreign examples results in a model that achieves a much lower error rate of 1.614%. As the model provides calibrated probabilistic predictions, it is also able to assess its confidence when making classifications. Among examples in which the model is confident (approximately 95% of the test data), the Bayesian DDM classifier achieves 0.45% error. Although these are not state-of-the-art rates, they show the flexibility of classifier construction in the DDM setting and how the normalized density can be leveraged *across* separately trained models, something not typically possible for energy-based probabilistic approaches.

We can extend these ideas further to use the deep density model even if only a small fraction of the observations are labelled. That is, we can use density estimates to extract useful information from unlabeled data by leveraging our knowledge of the empirical density in the representation space. One possible approach is to run the expectation-maximization algorithm and train the DDM on weighted data as part of a mixture model.

Appendix A

Proofs for Exact Recovery of PCA by Nested Dropout

Theorem 4. *Every optimal solution of the nested dropout problem is necessarily an optimal solution of the standard autoencoder problem.*

Proof. Let the nested dropout autoencoder be of latent dimension K. Recall that the nested dropout objective function in Equation (3.11) is a strictly positive mixture of the K different b-truncation problems. As described in Subsection 3.3.1, an optimal solution to each b-truncation must be of the form $\boldsymbol{X}_b^* = \boldsymbol{T}_b \Sigma_{\downarrow b} \boldsymbol{R}^T, \Gamma_b^* = \boldsymbol{Q}_{\downarrow b} \boldsymbol{T}_b^{-1}$ for some invertible transformation $\boldsymbol{T}_b$. We note that the PCA decomposition is a particular optimal solution for each b that is given for the choice $\boldsymbol{T}_b = \mathbb{I}_b$. As such, the PCA decomposition exactly minimizes *every* term in the nested dropout mixture, and therefore must be a global solution of the nested dropout problem. This means that every optimal solution of the nested dropout problem must exactly minimize every term in the nested dropout mixture. In particular, one of these terms corresponds to the K-truncation problem, which is in fact the original autoencoder problem. ■

Denote $\boldsymbol{T}_{\downarrow b} = \boldsymbol{J}_{K \to b} \boldsymbol{T} \boldsymbol{J}_{K \to b}^T$ as the b-th leading principal minor and its its bottom right corner as $t_b = T_{bb}$.

Lemma 2. *Let $\boldsymbol{T} \in \mathbb{R}^{K \times K}$ be commutative in its truncation and inversion. Then all the diagonal elements of $\boldsymbol{T}$ are nonzero, and for each $b = 2, \ldots, K$, either $\boldsymbol{A}_b = \boldsymbol{0}$ or*

$\boldsymbol{B}_b = \boldsymbol{0}$.

Proof. We have $\det \boldsymbol{T}_{\downarrow b} = \det \boldsymbol{T}_{\downarrow b-1} \det(t_b - \boldsymbol{B}_b \boldsymbol{T}_{\downarrow b-1}^{-1} \boldsymbol{A}_b) \neq 0$ since $\boldsymbol{T}_{\downarrow b-1}$ is invertible. Since $\boldsymbol{T}_{\downarrow b-1}$ is also invertible, then $t_b - \boldsymbol{B}_b \boldsymbol{T}_{\downarrow b-1}^{-1} \boldsymbol{A}_b \neq 0$. As such, we write $\boldsymbol{T}_{\downarrow b}$ in terms of blocks $\boldsymbol{T}_{\downarrow b-1}, \boldsymbol{A}_b, \boldsymbol{B}_b, t_b$, and apply blockwise matrix inversion to find that $\boldsymbol{T}_{\downarrow b-1}^{-1} = \boldsymbol{T}_{\downarrow b-1}^{-1} + \boldsymbol{T}_{\downarrow b-1}^{-1} \boldsymbol{A}_b (t_b - \boldsymbol{B}_b \boldsymbol{T}_{\downarrow b-1}^{-1} \boldsymbol{A}_b)^{-1} \boldsymbol{B}_b \boldsymbol{T}_{\downarrow b-1}^{-1}$ which reduces to $\boldsymbol{A}_b \boldsymbol{B}_b = \boldsymbol{0}$. Now, assume by contradiction that $t_b = 0$. This means that either bottom row or the rightmost column of $\boldsymbol{T}_{\downarrow b}$ must be all zeros, which contradicts with the invertibility of $\boldsymbol{T}_{\downarrow b}$. ■

Theorem 5. *Every optimal solution of the nested dropout problem must be of the form*

$$\begin{aligned} \boldsymbol{X}^* &= \boldsymbol{T} \Sigma \boldsymbol{R}^T && \text{(A.1)} \\ \boldsymbol{\Gamma}^* &= \boldsymbol{Q} \boldsymbol{T}^{-1}, && \text{(A.2)} \end{aligned}$$

for some matrix $\boldsymbol{T} \in \mathbb{R}^{K \times K}$ that is commutative in its truncation and inversion.

Proof. Consider an optimal solution $\boldsymbol{X}^*, \boldsymbol{\Gamma}^*$ of the nested dropout problem. For *each* b-truncation, as established in the proof of Theorem 1, it must hold that

$$\begin{aligned} \boldsymbol{X}_b^* &= \boldsymbol{T}_b \boldsymbol{J}_{K \to b} \Sigma \boldsymbol{R}^T && \text{(A.3)} \\ \boldsymbol{\Gamma}_b^* &= \boldsymbol{Q} \boldsymbol{J}_{K \to b}^T \boldsymbol{T}_b^{-1}. && \text{(A.4)} \end{aligned}$$

However, it must also be true that $\boldsymbol{X}_b = \boldsymbol{X}_{\downarrow b}, \boldsymbol{\Gamma}_b = \boldsymbol{\Gamma}_{\downarrow b}$ by the definition of the nested dropout objective in Equation (3.11). The first equation thus gives that $\boldsymbol{T}_b \boldsymbol{J}_{K \to b} = \boldsymbol{J}_{K \to b} \boldsymbol{T}_K$, and therefore $\boldsymbol{T}_b = \boldsymbol{J}_{K \to b} \boldsymbol{T}_K \boldsymbol{J}_{K \to b}^T = \boldsymbol{T}_{\downarrow b}$. This establishes the fact that the optimal solution for each b-truncation problem simply draws the b-th leading principal minor from the *same* "global" matrix $\boldsymbol{T} := \boldsymbol{T}_K$. The second equation implies that for every b, it holds that $\boldsymbol{J}_{K \to b} \boldsymbol{T}^{-1} \boldsymbol{J}_{K \to b}^T = (\boldsymbol{J}_{K \to b} \boldsymbol{T} \boldsymbol{J}_{K \to b}^T)^{-1}$ and as such $\boldsymbol{T}$ is commutative in its truncation and inversion. ■

Theorem 6. *Under the orthonormality constraint $\boldsymbol{\Gamma}^T \boldsymbol{\Gamma} = \mathbb{I}_K$, there exists a unique optimal solution for the nested dropout problem, and this solution is exactly the set* *of the K top eigenvectors of the covariance of $\boldsymbol{Y}$, ordered by eigenvalue magnitude. Namely, $\boldsymbol{X}^* = \Sigma \boldsymbol{R}^T, \boldsymbol{\Gamma}^* = \boldsymbol{Q}$.*

Proof. The orthonormality constraint implies $(\boldsymbol{T}^{-1}\boldsymbol{Q})^T\boldsymbol{Q}\boldsymbol{T}^{-1} = \mathbb{I}_K$ which gives $\boldsymbol{T}^T = \boldsymbol{T}^{-1}$. Hence every row and every column must have unit norm. We also have have that for every $b = 1, \ldots, K$

$$\begin{aligned}
\boldsymbol{T}_{\downarrow b}^T &= (\boldsymbol{J}_{K\to b}\boldsymbol{T}\boldsymbol{J}_{K\to b}^T)^T && \text{(A.5)}\\
&= \boldsymbol{J}_{K\to b}\boldsymbol{T}^T\boldsymbol{J}_{K\to b}^T && \text{(A.6)}\\
&= \boldsymbol{J}_{K\to b}\boldsymbol{T}^{-1}\boldsymbol{J}_{K\to b}^T && \text{(A.7)}\\
&= (\boldsymbol{J}_{K\to b}\boldsymbol{T}\boldsymbol{J}_{K\to b}^T)^{-1} && \text{(A.8)}\\
&= \boldsymbol{T}_{\downarrow b}^{-1} && \text{(A.9)}
\end{aligned}$$

where in the last equation we applied Lemma 1 to Theorem 2. As such, every leading principal minor is also orthonormal. For the sake of contradiction, assume there exist some $m, n, m \neq n$ such that $T_{mn} \neq 0$. Without loss of generality assume $m < n$. Then $\sum_{p=1}^{n-1} T_{mp}^2 < 1$, but this violates the orthonormality of $\boldsymbol{T}_{n-1}$. Thus it must be that the diagonal elements of $\boldsymbol{T}$ are all identically 1, and therefore $\boldsymbol{T} = \mathbb{I}_K$. The result follows. ■

Appendix B

Implementation Details for Metric Learning with Adaptive Density Discrimination

B.1 Hyperparameter Tuning Specifications and Optimal Configurations

Here we describe in detail the hyperparameter search setups for the different experiments, and the optimal configuration for each.

For all models, we tune optimization hyperparameters consisting of learning rate and its annealing factor which we apply every epoch. We fix the momentum as 0.9 for all experiments. For the smaller datasets, we refresh our index every epoch, and for ImageNet Attributes every 1000 iterations.

For Magnet Loss, we additionally tune the separation margin α, the number of nearest clusters per minibatch M, the number of examples per cluster D, and the number of clusters per class K which we take to be the same for all classes (the examples per minibatch MD is upper-bounded by 48 due to memory constraints). Note that we permit the choices $M = D = 2$, which, as discussed in 4.3.4, reverts this back to triplet loss: hence, we expect this choice to be discovered if triplet loss is in fact the optimal choice of distance metric learning loss of this class. For triplet loss, we tune the separation margin α, the fraction of nearest impostors retrieved in each minibatch and neighbourhood size retrieved for kNN evaluation.

We now specify the optimal hyperparameter configurations for the different datasets and model spaces, as found empirically via random search. The learning rate annealing factor is marked as "N/A" for smaller datasets, where we do not anneal the learning rate at all.

Model	**Hyperparameter**	**Pet**	**Flowers**	**Dogs**	**ImageNet Attributes**	**Hierarchy recovery**
Magnet	Learning rate	0.00184	0.0240	0.00292	0.00459	0.00177
	Annealing factor	N/A	N/A	N/A	0.974	0.988
	Gap	7.18	2.43	0.710	0.700	0.783
	Global scaling	3.52	14.2	3.03	6.42	2.33
	Clusters/class	8	1	1	2	16
Triplet	Learning rate	0.000598	0.00155	0.00293	0.00807	0.00187
	Annealing factor	N/A	N/A	N/A	0.966	0.995
	Gap	0.304	0.554	0.370	0.495	0.556
	Nearest impostor fraction	0.184	0.129	0.00713	0.0700	0.0424
	Neighbourhood size	128	128	128	128	128

Table B.1: Optimal hyperparameter configurations for the different datasets and model spaces.

B.2 Specifications for ImageNet Attributes Dataset

To curate this dataset, we first matched the annotated examples in the Object Attributes dataset (Russakovsky & Fei-Fei, 2010) to examples in the training set of ImageNet. The ImageNet Attributes training and validation sets then comprise all examples of all classes for which annotated examples exist.

Below we list these classes.

```
n01693334, n01773549, n01773797, n01796340, n01872401, n01873310, n01882714, n01883070,
n02071294, n02074367, n02088238, n02088364, n02088466, n02088632, n02090379, n02091134,
n02091635, n02092002, n02096294, n02100583, n02100735, n02101556, n02102480, n02104029,
n02104365, n02105056, n02105162, n02105251, n02105505, n02106030, n02109047, n02109525,
n02110806, n02110958, n02112350, n02115913, n02119789, n02123045, n02123394, n02124075,
n02125311, n02128925, n02129165, n02130308, n02326432, n02342885, n02361337, n02391049,
n02410509, n02422106, n02422699, n02423022, n02441942, n02442845, n02443114, n02443484,
n02444819, n02445715, n02447366, n02480495, n02480855, n02481823, n02483708, n02484975,
n02486261, n02486410, n02487347, n02488291, n02488702, n02500267, n02509815, n02536864,
```

n02802426 , n02808440 , n02910353 , n03249569 , n03325584 , n03721384 , n03977966 , n03982430 , n04118776 , n04228054 , n04447861 , n07615774 , n07745940 , n07873807 , n07875152 , n07880968 , n11939491 , n12267677

Appendix C

Implementation Details for Spectral Pooling

Here we provide additional detail pertaining to the specific algorithmic implementation of the spectral pooling and spectral parameterization. CropSpectrum and PadSpectrum are self-explanatory: they crop or zero-pad the frequency spectrum to the appropriate dimensionalities, respectively.

Algorithm 3 TreatCornerCases

```
Input: Input map $\mathbf{y} \in \mathbb{C}^{M \times N}$
Output: Output map $\mathbf{z}$ with corner cases obeying conjugate symmetry, special case
    indices $S$
 1: $\mathbf{z} \leftarrow \mathbf{y}$
 2: $S \leftarrow \{(0,0)\}$
 3: if $M$ is even then
 4:     $S \leftarrow \{(\frac{M}{2},0)\}$
 5: end if
 6: if $N$ is even then
 7:     $S \leftarrow \{(0,\frac{N}{2})\}$
 8: end if
 9: if $M$ is even and $N$ is even then
10:     $S \leftarrow \{(\frac{M}{2},\frac{N}{2})\}$
11: end if
12: for $i \in S$ do
13:     $\mathrm{Im}(\mathrm{z}_i) \leftarrow 0$
14: end for
```

Algorithm 4 RemoveRedundancy

```
Input: Input gradient map $\mathbf{y} \in \mathbb{C}^{M \times N}$
Output: Gradient $\mathbf{z}$ in terms of unconstrained parameters only
 1: $\mathbf{z}, S \leftarrow$ TreatCornerCases($\mathbf{y}$)
 2: $I \leftarrow \emptyset$
 3: for $m = 0, \ldots, M-1$ do
 4:   for $n = 0, \ldots, \lfloor \frac{N}{2} \rfloor$ do
 5:     if $(m, n) \notin S$ then
 6:       if $(m, n) \notin I$ then
 7:         $\mathrm{z}_{m,n} \leftarrow 2\mathrm{z}_{m,n}$
 8:         $I \leftarrow I \cup \{(m,n), ((M-m) \bmod M, (N-n) \bmod N)\}$
 9:       else
10:         $\mathrm{z}_{m,n} \leftarrow 0$
11:       end if
12:     end if
13:   end for
14: end for
```

Algorithm 5 RecoverMap

```
Input: Input gradient $\mathbf{y} \in \mathbb{C}^{M \times N}$ parametrized by unconstrained elements only
Output: Full gradient $\mathbf{z}$ with recovered redundancy
 1: $\mathbf{z}, S \leftarrow$ TreatCornerCases($\mathbf{y}$)
 2: $I \leftarrow \emptyset$
 3: for $m = 0, \ldots, M-1$ do
 4:   for $n = 0, \ldots, \lfloor \frac{N}{2} \rfloor$ do
 5:     if $(m, n) \notin S$ then
 6:       if $(m, n) \notin I$ then
 7:         $\mathrm{z}_{m,n} \leftarrow \frac{1}{2}\mathrm{z}_{m,n}$
 8:         $\mathrm{z}_{(M-m) \bmod M, (N-n) \bmod N} \leftarrow \mathrm{z}_{m,n}$
 9:         $I \leftarrow I \cup \{(m,n), ((M-m) \bmod M, (N-n) \bmod N)\}$
10:       else
11:         $\mathrm{z}_{m,n} \leftarrow 0$
12:       end if
13:     end if
14:   end for
15: end for
```

Appendix D

Implementation Details for Scalable Bayesian Optimization Using Deep Neural Networks

D.1 Convolutional neural network experiment specifications

In this section we elaborate on the details of the network architecture, training and the meta-optimization. In the following subsections we elaborate on the hyperparametrization scheme. The priors on the hyperparameters as well as their optimal configurations for the two datasets can be found in Table D.2.

D.1.1 Architecture

The model architecture is specified in Table D.1.

D.1.2 Data augmentation

We corrupt each input in a number of ways. Below we describe our parametrization of these corruptions.

Layer type	**# Filters**	**Window**	**Stride**
Convolution	96	3×3	
Convolution	96	3×3	
Max pooling		3×3	2
Convolution	192	3×3	
Convolution	192	3×3	
Convolution	192	3×3	
Max pooling		3×3	2
Convolution	192	3×3	
Convolution	192	1×1	
Convolution	10/100	1×1	
Global averaging		6×6	
Softmax			

Table D.1: Our convolutional neural network architecture. This choice was chosen to be maximally generic. Each convolution layer is followed by a ReLU nonlinearity.

HSV We shift the hue, saturation and value fields of each input by global constants $b_H \sim U(-B_H, B_H)$, $b_S \sim U(-B_S, B_S)$, $b_V \sim U(-B_V, B_V)$. Similarly, we globally stretch the saturation and value fields by global constants $a_S \sim U(\frac{1}{1+A_S}, 1 + A_S)$, $a_V \sim U(\frac{1}{1+A_V}, 1 + A_V)$.

SCALINGS Each input is scaled by some factor $s \sim U(\frac{1}{1+S}, 1 + S)$.

TRANSLATIONS We crop each input to size 27×27, where the window is chosen randomly and uniformly.

HORIZONTAL REFLECTIONS Each input is reflected horizontally with a probability of 0.5.

PIXEL DROPOUT Each input element is dropped independently and identically with some random probability D_0.

D.1.3 Initialization and training procedure

We initialize the weights of each convolution layer m with i.i.d zero-mean Gaussians with standard deviation $\frac{\sigma}{\sqrt{F_m}}$ where F_m is the number of parameters per filter for that layer. We chose this parametrization to produce activations whose variances are invariant to filter dimensionality. We use the same standard deviation for all layers but the input, for which we dedicate its own hyperparameter σ_I as it oftentimes varies in scale from deeper layers in the network.

We train the model using the standard stochastic gradient descent and momentum optimizer. We use minibatch size of 128, and tune the momentum and learning rate, which we parametrize as $1 - 0.1^M$ and 0.1^L respectively. We anneal the learning rate by a factor of 0.1 at epochs 130 and 190. We terminate the training after 200 epochs.

We regularize the weights of all layers with weight decay coefficient W. We apply dropout on the outputs of the max pooling layers, and tune these rates D_1, D_2 separately.

D.1.4 Testing procedure

We evaluate the performance of the learned model by averaging its log-probability predictions on 100 samples drawn from the input corruption distribution, with masks drawn from the unit dropout distribution.

D.2 Multimodal neural language model hyperparameters

D.2.1 Description of the hyperparameters

We optimize a total of 11 hyperparameters of the log-bilinear model (LBL). Below we explain what these hyperparameters refer to.

MODEL The LBL model has two variants, an additive model where the image features are incorporated via an additive bias term, and a multiplicative that uses a factored weight tensor to control the interaction between modalities.

CONTEXT SIZE The goal of the LBL is to predict the next word given a sequence of words. The context size dictates the number of words in this sequence.

LEARNING RATE, MOMENTUM, BATCH SIZE These are optimization parameters used during stochastic gradient learning of the LBL model parameters. The optimization over learning rate is carried out in log-space, but the proposed learning rate is exponentiated before being passed to the training procedure.

HIDDEN LAYER SIZE This controls the size of the joint hidden representation for words and images.

EMBEDDING SIZE Words are represented by feature embeddings rather than one-hot vectors. This is the dimensionality of the embedding.

DROPOUT A regularization parameter that determines the amount of dropout to be added to the hidden layer.

CONTEXT DECAY, WORD DECAY $\mathcal{L}_2$ regularization on the input and output weights respectively. Like the learning rate, these are optimized in log-space as they vary over several orders of magnitude.

FACTORS The rank of the weight tensor. Only relevant for the multiplicative model.

Hyperparameter	**Notation**	**Support of prior**	**CIFAR-10 Optimum**	**CIFAR-100 Optimum**
Momentum	M	$[0.5, 2]$	1.6242	1.3339
Learning rate	L	$[1, 4]$	2.7773	2.1205
Initialization deviation	σ_I	$[0.5, 1.5]$	0.83359	1.5570
Input initialization deviation	σ	$[0.01, 1]$	0.025370	0.13556
Hue shift	B_H	$[0, 45]$	31.992	19.282
Saturation scale	A_S	$[0, 0.5]$	0.31640	0.30780
Saturation shift	B_S	$[0, 0.5]$	0.10546	0.14695
Value scale	A_S	$[0, 0.5]$	0.13671	0.13668
Value shift	B_S	$[0, 0.5]$	0.24140	0.010960
Pixel dropout	D_0	$[0, 0.3]$	0.19921	0.00056598
Scaling	S	$[0, 0.3]$	0.24140	0.12463
L2 weight decay	W	$[2, 5]$	4.2734	3.1133
Dropout 1	D_1	$[0, 0.7]$	0.082031	0.081494
Dropout 2	D_2	$[0, 0.7]$	0.67265	0.38364

Table D.2: Specification of the hyperparametrization scheme, and optimal hyperparameter configurations found.

Hyperparameter	**Support of prior**	**Notes**	**COCO Optimum**
Model	$\{$additive,multiplicative$\}$		additive
Context size	$[3, 25]$		5
Learning rate	$[0.001, 10]$	Log-space	0.43193
Momentum	$[0, 0.9]$		0.23269
Batch size	$[20, 200]$		40
Hidden layer size	$[100, 2000]$		441
Embedding size	$\{50, 100, 200\}$		100
Dropout	$[0, 0.7]$		0.14847
Word decay	$[10^{-9}, 10^{-3}]$	Log-space	2.98456^{-7}
Context decay	$[10^{-9}, 10^{-3}]$	Log-space	1.09181^{-8}
Factors	$[50, 200]$	Multiplicative model only	-

Table D.3: Specification of the hyperparametrization scheme, and optimal hyperparameter configurations found for the multimodal neural language model. For parameters marked log-space, the log is given to the Bayesian optimization routine and the result is exponentiated before being passed into the multimodal neural language model for training. Square brackets denote a range of parameters, while curly braces denote a set of options.

Bibliography

Adams, Ryan P., Murray, Iain, and MacKay, David J. C. The Gaussian process density sampler. In *Advances in Neural Information Processing Systems 21*, 2009.

Adams, Ryan P., Wallach, Hanna M., and Ghahramani, Zoubin. Learning the structure of deep sparse graphical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

Angelova, Anelia and Long, Philip M. Benchmarking large-scale fine-grained categorization. In *IEEE Winter Conference on Applications of Computer Vision, Steamboat Springs, CO, USA, March 24-26, 2014*, pp. 532–539, 2014.

Angelova, Anelia and Zhu, Shenghuo. Efficient object detection and segmentation for fine-grained recognition. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pp. 811–818, 2013.

Arthur, David and Vassilvitskii, Sergei. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5.

Bardenet, Rémi, Brendel, Mátyás, Kégl, Balázs, and Sebag, Michèle. Collaborative hyperparameter tuning. In *ICML*, 2013.

Bell, A.J. and Sejnowski, T.J. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 7(6):1129–1159, 1995.

Bengio, Yoshua. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2 (1):1–127, January 2009. ISSN 1935-8237.

Bengio, Yoshua and LeCun, Yann. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines.* MIT Press, 2007.

Bengio, Yoshua, Louradour, Jérôme, Collobert, Ronan, and Weston, Jason. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009.

Bergstra, James and Bengio, Yoshua. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Bergstra, James S., Bardenet, Rémi, Bengio, Yoshua, and Kégl, Bálázs. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems.* 2011.

Bishop, Christopher M. *Pattern Recognition and Machine Learning.* Springer-Verlag New York, Inc., 2006.

Bourlard, H. and Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. Manuscript M217, Philips Research Laboratory, Brussels, Belgium, 1987.

Brochu, Eric, Brochu, Tyson, and de Freitas, Nando. A Bayesian interactive optimization approach to procedural animation design. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010.

Bruna, Joan and Mallat, Stephane. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

Bull, Adam D. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, (3-4):2879–2904, 2011.

Buntine, Wray L. and Weigend, Andreas S. Bayesian back-propagation. *Complex systems*, 5(6):603–643, 1991.

Calandra, Roberto, Peters, Jan, Rasmussen, Carl Edward, and Deisenroth, Marc Peter. Manifold Gaussian processes for regression. *preprint arXiv:1402.5876*, 2014a.

Calandra, Roberto, Peters, Jan, Seyfarth, Andre, and Deisenroth, Marc P. An experimental evaluation of Bayesian optimization on bipedal locomotion. In *International Conference on Robotics and Automation*, 2014b.

Carvalho, Carlos M., Polson, Nicholas G., and Scott, James G. Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, 2009.

Chopra, Sumit, Hadsell, Raia, and LeCun, Yann. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pp. 539–546, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2.

Coates, A., Lee, H., and Ng, A.Y. An analysis of single-layer networks in unsupervised feature learning. In *Proc. of AISTATS*, volume 15, pp. 215–223, 2011.

Cottrell, Garrison W., Munro, Paul, and Zipser, David. Learning internal representations from gray-scale images: An example of extensional programming. In *Conference of the Cognitive Science Society*, 1987.

Datar, Mayur, Immorlica, Nicole, Indyk, Piotr, and Mirrokni, Vahab S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pp. 253–262, New York, NY, USA, 2004. ACM.

De Freitas, Joao FG. *Bayesian methods for neural networks.* PhD thesis, Trinity College, University of Cambridge, 2003.

de Freitas, Nando, Smola, Alex J., and Zoghi, Masrour. Exponential regret bounds for Gaussian process bandits with deterministic observations. In *ICML*, 2012.

Djolonga, Josip, Krause, Andreas, and Cevher, Volkan. High dimensional Gaussian process bandits. In *Advances in Neural Information Processing Systems*, 2013.

Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. Decaf: A deep convolutional activation feature for generic visual recognition.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

Escobar, Michael D. and West, Mike. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, June 1995.

Feurer, M., Springenberg, T., and Hutter, F. Initializing Bayesian hyperparameter optimization via meta-learning. In *AAAI Conference on Artificial Intelligence*, 2015.

Garnett, Roman, Osborne, Micheal A., and Roberts, Stephen J. Bayesian optimization for sensor set selection. In *International Conference on Information Processing in Sensor Networks*, 2010.

Gavves, E., Fernando, B., Snoek, C. G. M., Smeulders, A. W. M., and Tuytelaars, T. Fine-grained categorization by alignments. In *IEEE International Conference on Computer Vision*, 2013.

Gavves, Efstratios, Fernando, Basura, Snoek, CeesG.M., Smeulders, ArnoldW.M., and Tuytelaars, Tinne. Local alignments for fine-grained categorization. *International Journal of Computer Vision*, 111(2):191–212, 2015. ISSN 0920-5691.

Gelbart, Michael A., Snoek, Jasper, and Adams, Ryan P. Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence*, 2014.

Ginsbourger, David and Riche, Rodolphe Le. Dealing with asynchronicity in parallel Gaussian process based global optimization. 2010.

Globerson, Amir and Roweis, Sam T. Metric learning by collapsing classes. In Weiss, Y., Schölkopf, B., and Platt, J.C. (eds.), *Advances in Neural Information Processing Systems 18*, pp. 451–458. MIT Press, 2006.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 513–520, 2011a.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. In *Proc. of AISTATS*, 2011b.

Goldberger, Jacob, Roweis, Sam, Hinton, Geoff, and Salakhutdinov, Ruslan. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, pp. 513–520. MIT Press, 2004.

Goodfellow, Ian J., Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *ICML*, 2013.

Gramacy, Robert B. and Lee, Herbert K. H. Optimization under unknown constraints, 2010. arXiv:1004.4027.

Grauman, Kristen and Fergus, Rob. Learning binary hash codes for large-scale image search. In *Machine Learning for Computer Vision*, volume 411 of *Studies in Computational Intelligence*, pp. 49–87. Springer Berlin Heidelberg, 2013.

Hadsell, Raia, Chopra, Sumit, and Lecun, Yann. Dimensionality reduction by learning an invariant mapping. In *In Proc. Computer Vision and Pattern Recognition Conference (CVPRâĂŹ06*. IEEE Press, 2006.

Hensman, J, Fusi, N, and Lawrence, N.D. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, 2013.

Hinton, G. E. and van Camp, D. Keeping neural networks simple by minimizing the description length of the weights. In *ACM Conference on Computational Learning Theory*, 1993.

Hinton, G.E., Osindero, S., and Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Hinton, Geoffrey. What's wrong with convolutional nets? *MIT Brain and Cognitive Sciences - Fall Colloquium Series*, Dec 2014a.

Hinton, Geoffrey. Ask me anything: Geoffrey hinton. *Reddit Machine Learning*, 2014b.

Hinton, Geoffrey and Salakhutdinov, Ruslan. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29 (6):82–97, 2012a.

Hinton, Geoffrey E. and Salakhutdinov, Ruslan. Using deep belief nets to learn covariance kernels for Gaussian processes. In *Advances in neural information processing systems*, pp. 1249–1256, 2008.

Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Improving neural networks by preventing co-adaptation of feature detectors. 2012b.

Hoffman, Matthew, Brochu, Eric, and de Freitas, Nando. Portfolio allocation for Bayesian optimization. In *Uncertainty in Artificial Intelligence*, 2011.

Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization 5*, 2011.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015a.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015b.

Jarrett, Kevin, Kavukcuoglu, Koray, Ranzato, Marc'Aurelio, and LeCun, Yann. What is the best multi-stage architecture for object recognition? In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pp. 2146–2153, 2009.

Jones, Donald R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21, 2001.

Karpathy, Andrej, Toderici, George, Shetty, Sanketh, Leung, Thomas, Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition*, 2014.

Khosla, Aditya, Jayadevaprakash, Nityananda, Yao, Bangpeng, and Fei-Fei, Li. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Kingma, Diederik P. and Welling, Max. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Kiros, Ryan, Salakhutdinov, Ruslan, and Zemel, Richard S. Multimodal neural language models. In *ICML*, 2014.

Krause, Andreas and Ong, Cheng Soon. Contextual Gaussian process bandit optimization. In *Advances in Neural Information Processing Systems*, 2011.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, Alex and Hinton, Geoffrey E. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.

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

Kushner, H. J. A new method for locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86, 1964.

Lawrence, N. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005.

Lawrence, Neil D. and Candela, Joaquin Quiñonero. Local distance preservation in the GP-LVM through back constraints. In *International Conference on Machine Learning*, pp. 513–520, 2006.

Lázaro-Gredilla, Miguel and Figueiras-Vidal, Aníbal R. Marginalized neural network mixtures for large-scale regression. *Neural Networks, IEEE Transactions on*, 21(8): 1345–1351, 2010.

LeCun, Yann and Bengio, Yoshua. *Convolutional Networks for Images, Speech and Time Series*, pp. 255–258. The MIT Press, 1995.

LeCun, Yann, Boser, Bernhard, Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel., L. D. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, 1989.

LeCun, Yann, Denker, John S., and Solla, Sara A. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 598–605, 1990.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521 (7553):436–444, 2015.

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. *CoRR*, abs/1409.5185, 2014.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *CoRR*, abs/1312.4400, 2013.

Lin, Tsung-Yi, Maire, Michael, Belongie, Serge, Hays, James, Perona, Pietro, Ramanan, Deva, Dollár, Piotr, and Zitnick, C Lawrence. Microsoft COCO: Common objects in context. In *ECCV 2014*, pp. 740–755. Springer, 2014.

Lizotte, Dan. *Practical Bayesian Optimization.* PhD thesis, University of Alberta, Edmonton, Alberta, 2008.

MacKay, David J.C. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

MacKay, David J.C. and Gibbs, Mark N. Density networks. In *Proceedings of Society for General Microbiology Edinburgh meeting*, 1997.

Mahendran, Nimalan, Wang, Ziyu, Hamze, Firas, and de Freitas, Nando. Adaptive MCMC with Bayesian optimization. In *Artificial Intelligence and Statistics*, 2012.

Mathieu, Michaël, Henaff, Mikael, and LeCun, Yann. Fast training of convolutional networks through FFTs. *CoRR*, abs/1312.5851, 2013.

Mensink, Thomas, Verbeek, Jakob, Perronnin, Florent, and Csurka, Gabriela. Distance-based image classification: Generalizing to new classes at near zero cost. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2013.

Min, Martin Renqiang, van der Maaten, Laurens, Yuan, Zineng, Bonner, Anthony J., and Zhang, Zhaolei. Deep supervised t-distributed embedding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 791–798, 2010.

Mnih, Andriy and Gregor, Karol. Neural variational inference and learning in belief networks. In *ICML*, 2014.

Mockus, Jonas, Tiesis, Vytautas, and Zilinskas, Antanas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2, 1978.

Murray, Naila and Perronnin, Florent. Generalized max pooling. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pp. 2473–2480, 2014.

Neal, Radford. Slice sampling. *Annals of Statistics*, 31:705–767, 2000.

Neal, Radford M. Connectionist learning in belief networks. *Artificial Intelligence*, 56:71–113, July 1992.

Neal, Radford M. *Bayesian learning for neural networks.* PhD thesis, University of Toronto, 1995.

Nickson, Thomas, Osborne, Michael A., Reece, Steven, and Roberts, Stephen. Automated machine learning using stochastic algorithm tuning. *NIPS Workshop on Bayesian Optimization*, 2014.

Nilsback, M-E. and Zisserman, A. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

Norouzi, Mohammad, Fleet, David, and Salakhutdinov, Ruslan R. Hamming distance metric learning. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1061–1069. Curran Associates, Inc., 2012.

Osborne, Michael A., Garnett, Roman, and Roberts, Stephen J. Gaussian processes for global optimization. In *Learning and Intelligent Optimization*, 2009.

Ouyang, Wanli and Wang, Xiaogang. Joint deep learning for pedestrian detection. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

Oyallon, Edouard, Mallat, Stéphane, and Sifre, Laurent. Generic deep networks with wavelet scattering. *CoRR*, abs/1312.5940, 2013.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988.

Qian, Qi, Jin, Rong, Zhu, Shenghuo, and Lin, Yuanqing. Fine-grained visual categorization via multi-stage metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Rasmussen, Carl Edward. The infinite Gaussian mixture model. In *Advances in Neural Information Processing Systems 12*, pp. 554–560, 2000.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and variational inference in deep latent Gaussian models. In *ICML*, 2014.

Rifai, Salah, Bengio, Yoshua, Dauphin, Yann, and Vincent, Pascal. A generative process for sampling contractive auto-encoders. In *International Conference on Machine Learning*, 2012.

Rippel, Oren and Adams, Ryan P. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

Rippel, Oren, Gelbart, Michael A., and Adams, Ryan P. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, 2014.

Rippel, Oren, Snoek, Jasper, and Adams, Ryan P. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, 2015.

Rippel, Oren, Paluri, Manohar, Dollar, Piotr, and Bourdev, Lubomir. Metric learning with adaptive density discrimination. In *International Conference on Learning Representations*, 2016.

Roweis, S.T. and Saul, L.K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Russakovsky, Olga and Fei-Fei, Li. Attribute learning in large-scale datasets. In *European Conference of Computer Vision (ECCV), International Workshop on Parts and Attributes*, 2010.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, pp. 1–42, April 2015.

Salakhutdinov, Ruslan and Hinton, Geoffrey. Learning a nonlinear embedding by preserving class neighborhood structure. In *Proc. of AISTATS*, volume 11, 2007.

Salakhutdinov, Ruslan and Hinton, Geoffrey E. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.

Schölkopf, B., Smola, A., and Müller, K.R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Schroff, Florian, Kalenichenko, Dmitry, and Philbin, James. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Sharif Razavian, Ali, Azizpour, Hossein, Sullivan, Josephine, and Carlsson, Stefan. Cnn features off-the-shelf: An astounding baseline for recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. 1986.

Snelson, Edward and Ghahramani, Zoubin. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pp. 1257–1264, 2005.

Snoek, Jasper. *Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology.* PhD thesis, University of Toronto, Toronto, Canada, 2013.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan Prescott. Practical Bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

Snoek, Jasper, Swersky, Kevin, Zemel, Richard S., and Adams, Ryan P. Input warping for Bayesian optimization of non-stationary functions. In *ICML*, 2014.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Md. Mostofa Ali, Prabhat, and Adams, Ryan P. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin A. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

Srinivas, Niranjan, Krause, Andreas, Kakade, Sham, and Seeger, Matthias. Gaussian process optimization in the bandit setting: no regret and experimental design. In *ICML*, 2010.

Swersky, Kevin, Snoek, Jasper, and Adams, Ryan Prescott. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, 2013.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *CVPR 2015*, 2015.

Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, and Wolf, Lior. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1701–1708, 2014.

Tenenbaum, J.B., De Silva, V., and Langford, J.C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Titsias, M. and Lawrence, N. Bayesian Gaussian process latent variable model. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Titsias, Michalis K. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pp. 567–574, 2009.

Torralba, Antonio and Oliva, Aude. Statistics of natural image categories. *Network*, 14(3):391–412, August 2003. ISSN 0954-898X.

Torralba, Antonio, Fergus, Robert, and Freeman, William T. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

van der Maaten, L.J.P. and Hinton, G.E. Visualizing high-dimensional data using t-sne. 2008.

Vasilache, Nicolas, Johnson, Jeff, Mathieu, Michaël, Chintala, Soumith, Piantino, Serkan, and LeCun, Yann. Fast convolutional nets with fbfft: A GPU performance evaluation. *CoRR*, abs/1412.7580, 2014.

Verma, Nakul, Mahajan, Dhruv, Sellamanickam, Sundararajan, and Nair, Vinod. Learning hierarchical similarity metrics. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2280–2287. IEEE, 2012.

Vincent, Pascal and Bengio, Yoshua. Manifold Parzen windows. In *Advances in Neural Information Processing Systems 15*, pp. 825–832. MIT Press, 2002.

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 1096–1103, 2008.

Vincent, Pascal, Larochelle, Hugo, Lajoie, Isabelle, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

Walder, Christian and Schölkopf, Bernhard. Diffeomorphic dimensionality reduction. In *Advances in Neural Information Processing Systems 22*, 2008.

Wan, Li, Zeiler, Matthew D., Zhang, Sixin, LeCun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *ICML*, 2013.

Wang, Jiang, Song, Yang, Leung, Thomas, Rosenberg, Chuck, Wang, Jingbin, Philbin, James, Chen, Bo, and Wu, Ying. Learning fine-grained image similarity with deep ranking. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1386–1393. IEEE, 2014.

Wang, Ziyu, Zoghi, Masrour, Hutter, Frank, Matheson, David, and de Freitas, Nando. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, 2013.

Weinberger, Kilian Q. and Saul, Lawrence K. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, June 2009. ISSN 1532-4435.

Williams, Christoper K. I. Computing with infinite networks. In *Advances in Neural Information Processing Systems*, 1996.

Xie, Saining, Yang, Tianbao, Wang, Xiaoyu, and Lin, Yuanqing. Hyper-class augmented and regularized deep learning for fine-grained image classification. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 2645–2654, 2015.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhutdinov, Ruslan, Zemel, Richard, and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044v2*, 2015.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *arXiv preprint arXiv:1207.0580*, 2015.

Zeiler, Matthew D. and Fergus, Rob. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.