AN ABSTRACT OF THE THESIS OF

Neale Ratzlaff for the degree of Master of Science in Computer Science presented on November 20, 2018.

Title: Methods for Detection and Recovery of Out-of-Distribution Examples

Abstract approved: _____

Fuxin Li

Deep neural networks currently comprise the backbone of many applications where safety is a critical concern, for example: autonomous driving and medical diagnostics. Unfortunately these systems currently fail to detect out-of-distribution (OOD) inputs and can be prone to making dangerous errors when exposed to them. In addition, these same systems are vulnerable to maliciously altered inputs called adversarial examples. In response to these problems we present two methods to handle out-of-distribution inputs, as well resist adversarial examples, respectively.

To detect OOD inputs, we introduce HyperGAN: a generative adversarial network which learns to generate all the parameters of a deep neural network. HyperGAN first transforms low dimensional noise into a latent space, which can be sampled from to obtain diverse, performant sets of parameters for a target architecture. By sampling many sets of parameters, we form a diverse ensemble which provides a better estimate of uncertainty than standard ensembles. We show that HyperGAN can reliably detect OOD inputs as well as adversarial examples.

We also present a method for recovering clean images from adversarial examples. BFNet uses a differentiable bilateral filter as a preprocessor to a neural network. The bilateral filter projects inputs back to the space of natural images, and in doing so it removes the adversarial perturbation. We show that BFNet is an effective defense in multiple attack settings, and is able to provide additional robustness when combined with other defenses.

Methods for Detection and Recovery of Out-of-Distribution Examples

by

Neale Ratzlaff

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 20, 2018
Commencement June 2019

Master of Science thesis of Neale Ratzlaff presented on November 20, 2018.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Neale Ratzlaff, Author

ACKNOWLEDGEMENTS

I would like to sincerely thank those who believed in me enough to encourage me to keep learning. I would also like to thank my friends and mentors who have always encouraged me to remember what we aim for, and what we risk if we fail.

"The shadow of that hyddeous strength, sax myle and more it is of length".
David Lyndsay. *The Monarchie* (1554)

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDIX FIGURES

Figure Page

Chapter 1: Introduction

1.1 Overview

Recent advances in machine learning have yielded enormous value in terms of the performance of supervised learning methods. Deep neural networks (DNNs) in particular have shown incredible success on many difficult tasks such as image recognition, machine translation, density estimation, and many others. This success has spawned an exponential increase in research on neural network based predictive models. The number of research articles submitted to conferences which use neural networks has exploded since 2012. This has set the precedent whereby neural networks are added to any method that makes predictions.

1.1.1 Neural Networks

Neural networks at their most simplistic are used to approximate some arbitrary function f. For instance, a classifier $y = f(x)$ which maps some inputs x to their target categorical outputs $y \in \{0, 1, \ldots, n\}$. Neural networks typically consist of multiple sets of linear mappings $\phi(x)$, called *layers*, composed in a directed graph where the output of one layer is used as input to the next. The hierarchical architecture encourages each layer to learn an increasingly abstract representation of its input. Until the final layer where, in the classifier setting, the output of the last layer is mapped to the categorical label y.

The goal is to learn the mapping $\mathbf{y} = f(x; \theta)$, where each layer ϕ_i is parameterized by some weights θ_i. The full set of learned parameters θ represent the best possible approximation to f. The question of how to arrive at the best set of parameters has been answered in various ways through the years. Historically, the θ has taken one of three forms.

- $\phi_i(x; \theta_i)$ is a generic function, such as the RBF kernels used in kernel SVMs. These functions have enough parameters to memorize the dataset, but they generally do not generalize well.

- ϕ_i may also be manually set to reflect features which human designers believe to be conducive to approximating the true function f. This was the dominant method in computer vision for decades. Hand-designed functions such as Canny filters or wavelet transforms were common in segmentation and classification pipelines. Other domains such as natural language processing had their own heuristic functions. Each heuristic was task-specific and necessitated domain experts for individual applications.

- The current method for determining θ is to learn them through maximum likelihood. The parameters which minimize the given loss function are chosen as the best parameters for the model. The loss function gives an estimate of how well the current parameters perform on the data. The gradient of this loss function tells which direction we should update the parameters to further minimize the loss function. Given the gradient, the parameters of each layer are updated with the backpropagation algorithm. Learning the parameters means we are effectively searching a huge space of functions for the one that best approximates f. Because θ is learned, it may be very generic and have high capacity. It may also generalize well as we can find a family of functions which works well on all the input (training) data.

1.1.2 Gradient Descent and Maximum Likelihood

The optimization algorithm primarily employed in the training of neural networks is called Gradient Descent (GD). GD is commonly used when training sets are large. In theory, convex optimization algorithms converge exactly. However, nearly all neural networks have globally non-convex parameter spaces. Without guarantees on convexity, first or second order gradient information is purely local. In a strictly convex setting, gradients are global underestimators of the function values, allowing algorithms like gradient descent of have convergence guarantees as the (global) descent direction is always known. Because parameter spaces in neural networks are non-convex, a gradient only provides a local descent direction, revealing little information about the global structure of the space. Therefore, small steps according to the local gradient must be taken until some stopping point is reached. There are popular variants on GD such as

stochastic gradient descent (SGD), projected gradient descent (PGD), adaptive moment estimation (Adam), and many others. These algorithms do not fundamentally deviate from GD, so they will not impact our discussion. In a standard neural network, we search for parameters using the maximum likelihood principle. In short, under maximum likelihood we wish to find the parameters θ which maximize the log-likelihood of the data under the proposed model.

$$\theta_{ML} = \arg\max_{\theta} p_{model}(X;\theta) \tag{1.1}$$

$$= \arg\max_{\theta} \prod_{i=1}^{n} p_{model}(x_i;\theta) \tag{1.2}$$

$$= \arg\max_{\theta} \sum_{i=1}^{n} \log p_{model}(x_i;\theta) \tag{1.3}$$

$$= \arg\max_{\theta} \mathbb{E}_{x\sim p_{data}} \log p_{model}(x_i;\theta) \tag{1.4}$$

$$\tag{1.5}$$

We are finding θ such that the log-likelihood of the data is maximized. The log term of the final equation is known as the cost, or loss function of the model. Given a model, the loss function evaluates how well the model explains the input data. We can use the given cost as a proper scoring function, which we can use an objective, namely the negative log-likelihood, or NLL loss.

$$L(\theta) = -\mathbb{E}_{x,y\sim p_{data}} \log p_{model}(y|x) \tag{1.6}$$

Our objective in training neural networks is just the same. We arrive at our choice of θ by using GD. The gradient of any function points in the direction of steepest ascent. This fact helps us as we can use negative gradient of our cost function to find the direction to move θ towards the minimum. If the cost function is good, i.e. it provides an accurate estimation of how well the model explains the data, and if the training set is drawn I.I.D. from the true data distribution, then minimizing the cost function with gradient descent should yield a set of parameters which achieve minimal risk.

1.1.3 Implications of Maximum Likelihood

If we carefully consider the maximum likelihood principle we see that it guarantees remarkably little about the estimate of the parameters returned. Increasing the likelihood of the data does not necessarily go hand-in-hand with predictive capability. Models must be regularized effectively to keep them from overfitting to the training set. The maximum likelihood principle is satisfied by perfectly predicting the training set, while failing utterly on the test set. Especially in the case of deep (multi-layer) neural networks, which have more than enough to memorize every training example in a large dataset. Neural networks are especially prone to overfitting, and are in fact encouraged to do so. There are existing methods for model regularization such as L2/L1 weight penalties, dataset augmentation, or noise injection. These methods have been frequently used, but it's still an open question if any of these techniques work effectively on the incredibly high dimensional loss surfaces navigated by neural networks.

Overfitting in neural networks has been well controlled to the extent that they can perform very well on some leave-out test set, as well as the training set. This is the main goal of learning algorithms - to learn a representation which does well on both seen and unseen data. However, a test set is always limited; a test set can not possibly represent all conceivable inputs which a classifier might see. Hence, the ability of a neural network to generalize to inputs far from the training/testing distribution is neither tested nor considered. In a closed research setting this is acceptable. However, Neural networks now comprise the backbone of many budding industrial technologies such as autonomous cars, automatic medical diagnosis, machine translation, and others. These technologies must work in the world outside of the lab setting, and be equipped to handle inputs far from the training manifold. If we are interested in neural networks performing well on off-manifold data, then the simplest method by far is to simply sample more data and add it to the training set. We can immediately see that this strategy doesn't scale well. It is a never-ending endeavor to outfit an autonomous car with a classifier which can recognize all possible objects it could see. If we operate purely under the principle of maximum likelihood, we can never expect better than undefined behavior on new data.

1.1.4 Uncertainty

What behavior then do we expect from our ML systems? We can not form a training set of all objects in all orientations in all scenarios. Instead we should seek a simple modification to our predictions on data seen and unseen: uncertainty. Uncertainty is a measure of how likely the data is to have come from the training distribution, according to the predicting model. On inlier data, we expect a classifier to be confidant about both the likelihood and class of the data. On outlier data, we expect the classifier to be uniformly unconfident as to the category of the input. For example, let us consider a classifier trained on 10 different dog breeds. For each input, the classifier outputs a length-10 vector, where each entry corresponds to the likelihood that the inputs belongs to the indicated class. If the classifier input is an image of an inlier dog, we expect the 10 output probabilities to reflect an accurate estimation of how likely the input dog is to belong to each of the classes. If a Golden Retriever is one of the 10 dog classes, then an input featuring the mean golden retriever $p(Golden|\theta)$ should evaluate to a probability of $p = 1.0$ for the Golden Retriever class, and $p = 0.0$ for the other 9 categories. On the other hand, an input image of a car $p(Car|\theta)$ should correspond to uniform probability $p = 0.1$ for each of the 10 known classes, as the input can not be said to be one dog class any more than the others. Another way to say this is that we expect low entropy in the predictive distribution on inlier data, and high entropy otherwise.

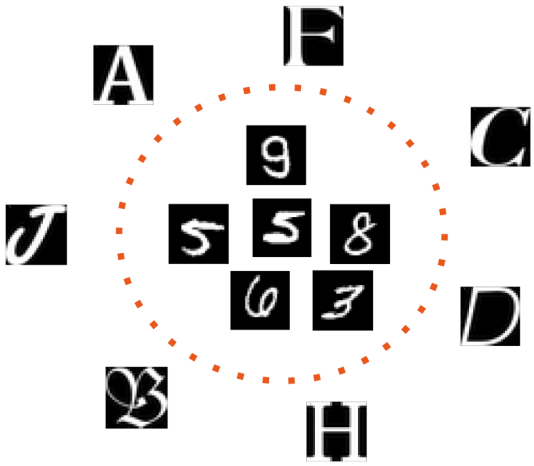


Figure 1.1: Example of out-of-domain vs in-domain inputs

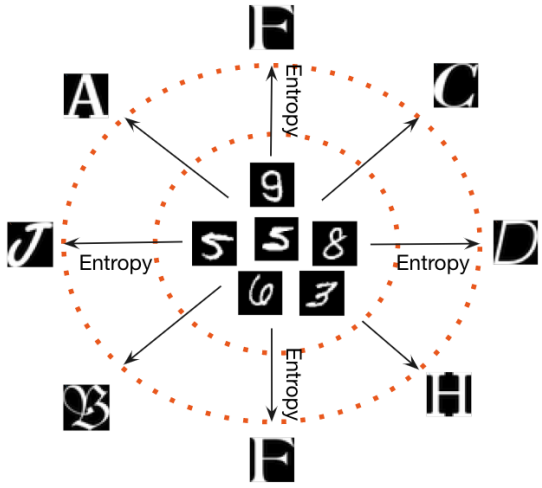


Figure 1.2: Entropy in predictions should increase as we move farther from the training manifold

There are many methods for coaxing uncertainty estimates from neural network predictions. They will be discussed further in chapter 2, along with a novel method that we propose here. To complete our introduction, we now turn to a class of OOD data which is fundamentally different from the unavoidable scenario of encountering real-world data that hasn't been seen before.

1.1.5 Adversarial Examples

The treatment of OOD data by standard neural networks is concerning. We want to rely on the predictions of the models we train and deploy. By contrast, there exist broad classes of input data which are expressly crafted to fool the model into giving incorrect outputs for data which looks like inlier data. Except for some esoteric methods, these *adversarial examples* consist of clean, inlier input data which has been modified to fool the predicting model. In most cases, these perturbations are calculated such that they represent the minimum change to an input needed to change the prediction. To illustrate this process, we will consider one of the first adversarial examples ever created. [47] showed that adversarial examples could be created by solving a simple optimization

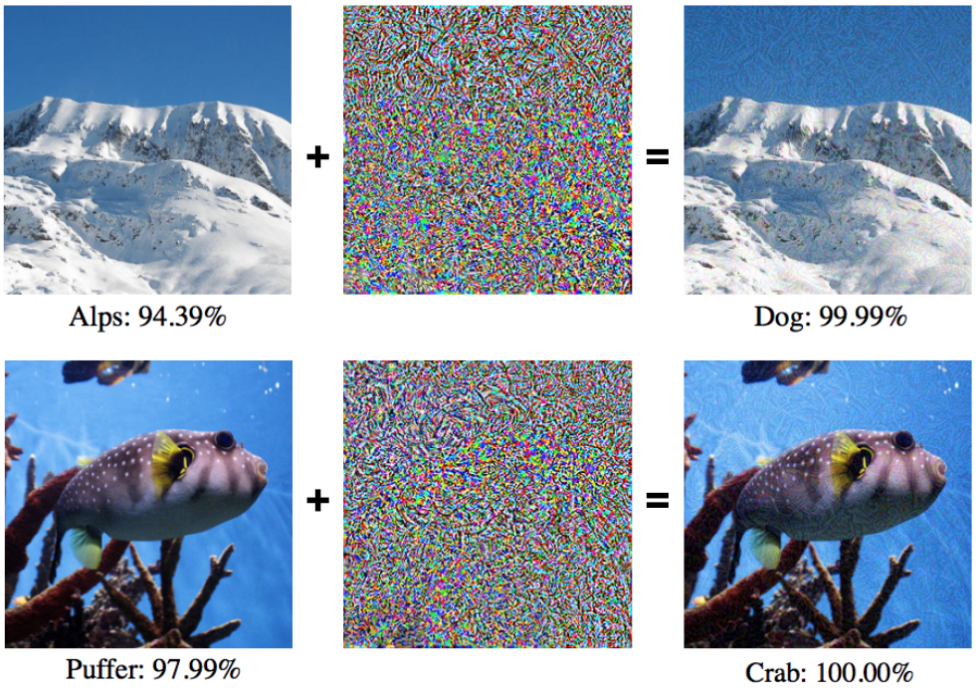


Figure 1.3: Visualizing the adversarial perturbations

problem.

$$\min \left[c \cdot ||\delta||_2^2 + L(x + \delta, y_{target}) \right]$$

The solution to the above problem satisfies two terms. The left term states that the distance between the original (clean) data x and the perturbed data $(x + \delta)$ should be small under the L2 norm. The second condition is that the perturbed data should be classified as some target class $y_{target} \neq y_{true}$. One would expect that changing an input such that it's classified differently, would require a meaningful change to high level features. Curiously, that is not the case; very small perturbations (imperceptible to humans) can be applied to an input image such that it's predicted incorrectly and with high confidence. This fact has enormous implications for the applicability of these systems in the outside world. If small perturbations can be added to objects such that they're misclassified, we can not trust an autonomous car with any degree of certainty. Stop signs may be green lights, pedestrians could be clear roads. To a vision system

using undefended neural networks, adversarial examples are a severe threat. There are many methods to create adversarial examples as well as defenses against them, all of which are detailed in chapter 3.

1.2 Contributions

We have now defined two areas of interest with regards to how neural networks make predictions.

- We expect a neural network to be uncertain when given OOD data as input. The class-probabilities given by the output of a classifier should reflect the probability that the input data came from the training distribution – or that the input data and training distribution have common support.
- Adversarial examples pose a clear threat to important computer vision systems. Defending against them, as well as understanding why neural networks are vulnerable is paramount.

In this thesis, we present methods for improving the performance of neural networks in both of these problematic domains. In chapter 2 we introduce HyperGAN: a generative model which learns a distribution over parameters. By predicting with many different sets of parameters, an estimate of uncertainty an be formed. we show that this method can safely detect OOD data as well as adversarial examples without incurring a large penalty on predictive accuracy. Note that HyperGAN may detect adversarial examples, but offers no guarantees on its robustness to them.

To mitigate the problem of adversarial examples, in chapter 3 we introduce BFNet. BFNet posits that adversarial examples lie off the training manifold, and uses the bilateral filter to project the data back onto the natural image manifold. BFNet does not need to know anything about the density of the training data (unlike HyperGAN). Instead it uses the fact that adversarial examples are created by perturbing inlier data. The bilateral filter thus removes the perturbations and recovers the original clean data.

Following these two proposed methods, we discuss briefly the implications of these methods, as well as the work left to do until these problems are solved.

Chapter 2: Detecting Out of Distribution Data

Since the inception of deep neural networks, it has been found that it is possible to train models from different random initializations and obtain networks that, albeit having quite different parameters, achieve quite similar accuracy [15]. It has further been found that ensembles of deep networks that are trained in such a way have significant performance advantages over single models [31], similar to the classical bagging approach in statistics. Ensemble models also have other benefits, such as being robust to outliers and being able to provide variance or uncertainty estimates over their inputs [27].

Standard training of neural networks cannot capture uncertainty, as the probability estimates obtained by applying the softmax function over the network outputs merely provides a maximum likelihood (or MAP) estimate given a single set of parameters. These estimates tend to be over-confident and thus are not capable of providing an accurate measurement of uncertainty. Neural networks are known to assign high predictive probability to data outside the training distribution, and relying on a point estimate of confidence is not enough to measure uncertainty

Past work has shown that deep networks are often over-parameterized [50, 4], having enough capacity to memorize entire datasets. Indeed one can sparsify deep convolutional networks (in some cases zeroing 90% of the weights) without losing significant accuracy [2]. We then hypothesize that the effective dimensionality of neural networks is not as large as is commonly presupposed. We then take into account these two facts: sparse networks can achieve similar accuracy as their dense counterparts, and that one can ensemble meaningfully different models trained from random initializations. Together, this leads us to the premise of our method: there exists a low-dimensional manifold of neural network parameters, where samples from this manifold correspond to models which achieve similarly good generalization accuracy on the same dataset.

2.1 Approaches to handle OOD data

2.1.1 Bayesian Deep Learning

Uncertainty in neural networks often requires a probabilistic model. Much of the literature concerning learning distributions over model space has been through the lens of Bayesian deep learning [30] [26]. Performing approximate Bayesian inference here is difficult, as integrating over the posterior (distribution of model parameters) is absolutely intractable. Instead a simple prior over both the data and the model parameters is assumed, and the prior is fit to an estimation of the true posterior. Variational inference (VI), Markov Chain Monte Carlo (MCMC), expectation propagation [35] or normalizing flows [41, 42, 25] are used to approximate the posterior instead. Notably, in addition to being generally harder to implement and requiring more training time than standard methods, a Bayesian approximation of uncertainty is highly dependant on the quality of the prior, which can often result in overfitting of the data if the prior is simple. Due to the maximum entropy principle of preferring a flexible prior, this is often true.

There has been recent work in obtaining uncertainty estimates from the variance in predictions of dropout models. Dropout can be viewed as a Bayesian approximation, integrating over the parameters of the network. Recently, [16] showed that networks with dropout following each layer are equivalent to a deep Gaussian process [9] marginalized over its covariance functions. They proposed MCdropout as a simple way to estimate model uncertainty. These approximations are not well aligned with current training patters of neural networks. Applying dropout to every layer results in over-regularization and underfitting of the target function. Moreover, dropout does not integrate over the full variation of possible models, only those which may be reached from (random) initialization.

2.1.2 Meta Learning

Meta learning approaches use different kinds of weights in order to increase the generalization ability of neural networks. The first proposed method is fast weights [23] which uses an auxiliary (slow) network to produce weight changes in the target (fast) network, acting as a short term memory store. Meta Networks [37] build on this approach by using an external neural memory store in addition to multiple sets of fast and slow weights. [6]

augment recurrent networks with fast weights as a more biologically plausible memory system than Neural Turing Machines. In the true spirit of meta learning, even optimizers can be formulated as auxiliary neural networks which predict updates to the weights of a target network [3]. At any step during training, the target weights are a function of the weights of the optimizing network. Our method can generate many different networks instead of learning to optimize just one, while remaining compatible with different losses and network architectures. At a high level, the meta learning approach of learning an abstraction over the weights of the target network is similar in nature to what we propose, with some key differences. Moreover, methods which employ fast weights form a tightly coupled system which cannot be separated.

2.1.3 Hypernetworks

As another interesting direction, hypernetworks [21] are neural networks which output parameters for a target neural network. The hypernetwork and the target network together form a single model which is trained jointly. The original hypernetwork produced the target weights as a deterministic function of its own weights, but Bayesian Hypernetworks (BHNs) [26], generate model parameters by sampling a Gaussian prior. Hypernetworks may serve as a framework to generate parameters for a target model, though they have not been used to directly generate full networks before.

2.1.4 Ensembles

Ensembles have long been used as a way to increase model performance [11], and also to give an uncertainty estimate on inputs [24]. Models which are trained from random initialization will thus reach different local minima through gradient descent. Assuming a population of diverse models, ensembling them and predicting according to a majority vote is one way to perform model averaging. The learned functions of each of the ensemble members are similar, despite being trained to different local minima. For this reason, we can perform model averaging without being in danger of drastically destroying model performance Here, model averaging has the relevant benefits of reducing effects of overfitting, as well as reducing variance. The failure of modern deep networks to generalize to OOD data may be viewed through a lens of overfitting, as networks have

more than enough parameters to memorize the domain they are trained on. Building on standard ensembles, [27] recently proposed Deep Ensembles, where adversarial training was applied to the training of individual experts in an ensemble in order to smooth the predictive variance even more. However, adversarial training is a very expensive training process, where adversarial examples must be generated for each batch of data seen. We seek a method to learn a distribution over parameters which does not require adversarial training.

2.2 HyperGAN

2.2.1 Overview

In this paper we explore an approach which focuses on generating all the parameters of a neural network, without assuming any fixed noise models on parameters. To keep our method scalable, we avoid utilizing invertible functions as in Bayesian approaches, and instead utilize the ideas from generative adversarial networks (GANs). We especially observe recent adversarial autoencoder [33] approaches. These approaches have demonstrated an impressive capability to model complicated, multimodal distributions in an unsupervised manner. In our approach, a random noise vector is first mixed into a number of different random vectors, and then each random vector serves as an embedding from which we generate all parameters within one layer of a deep network. The generator is then trained with conventional maximum likelihood (classification/regression) on the parameters it generates. To keep the embeddings from collapsing to a single mode, we employ adversarial regularization on the embeddings. In this way, it is possible to generate much larger networks than the dimensionality of the latent code, making our approach capable of generating all the parameters of a deep network with a single GPU. As an example, in our experiments on CIFAR-10 we start from a 256-dimensional latent vector and generate all $50,000+$ parameters in one pass, consuming only 4GB GPU memory. This shows that deep networks may indeed span a low-dimensional manifold, and could spur further thoughts and research.

2.2.2 Summary of Contributions

We propose HyperGAN, a novel approach for generating all the parameters for a target network architecture using a modified GAN, and we do so starting from a small Gaussian noise vector which scales well with the size of the output. Our approach is different from Bayesian approaches since we do not attempt to model the entire posterior. After our GAN is trained, one can directly generate many diverse, well-trained deep models without needing to further train or fine-tune them. The diversity of the models we can generate is beyond just adding dropout or scaling factors, which is shown by the superior performance of ensembles of the generated networks.

We believe HyperGAN is widely applicable to a variety of tasks, including meta-learning and reinforcement learning. One area where populations of diverse networks show promise is in uncertainty estimation and anomaly detection. We show through a variety of experiments that populations of networks sampled from HyperGAN are able to approximate the data distribution such that it can detect out of distribution samples. We show that we can provide a reasonable measure of uncertainty by calculating the entropy within the predictive distribution of sampled networks. Our method is straightforward, as well as easy to train and sample from. We hope that we can inspire future work in estimation of the manifold of neural networks.

2.3 HyperGAN

Taking note from the original hypernetwork framework for generating neural networks [21], we coin our approach HyperGAN. Standard GAN training dictates that we train a generator G to model the target distribution, aided by a discriminator network D. This entails acquiring a large dataset of trained neural network parameters, and providing samples to D to judge the output of the generator. However, a large collection of neural networks would be extremely costly to build. Instead, our data distribution is built online by evaluating the performance of our generated parameters at each step.

Taking note from the original hypernetwork framework for generating neural networks [21], we coin our approach HyperGAN. Standard GAN training dictates that we train a generator G to model the target distribution, aided by a discriminator network D. This entails acquiring a large dataset of trained neural network parameters, and providing

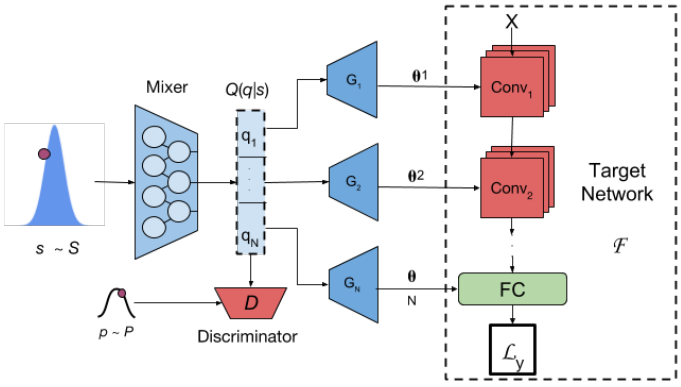


Figure 2.1: HyperGAN architecture. The mixer transforms $s \sim \mathcal{S}$ into latent codes $\{q_1, \ldots, q_N\}$. The generators each transform a latent subvector q_i into the parameters of the corresponding layer in the target network. The discriminator forces $Q(q|s)$ to be well-distributed and close to $\mathcal{P}$

samples to D to judge the output of the generator. However, a large collection of neural networks would be extremely costly to build. Instead, our data distribution is built online by evaluating the performance of our generated parameters at each step.

We begin with assuming that a neural network $\mathcal{F}(x;\theta)$ with input x and parameters θ consisting of a given architecture with N layers, and a training set M with inputs and targets $(X, Y) = \{x_i, y_i\}_{i=1}^{M}$. The standard training regime consists of computing a loss function $\mathcal{L}(\mathcal{F}(x;\theta), y)$ and updating the parameters θ with backpropagation until $\mathcal{L}$ is minimized. This works if we only want a point estimate of θ. However, if we want to generate more than one non-trivial network, some diversity is needed. Our approach to creating diversity is similar to a regular GAN architecture in that we start by drawing a random sample $s \sim \mathcal{S} = \mathcal{N}(\mathbf{0}, \mathbf{I}_j)$, where 0 is an all-zero vector and $\mathbf{I}_j$ is a $j \times j$ identity matrix, and generate parameters using a generator from s.

Figure 2.1 shows the HyperGAN architecture. Distinct from the standard GAN, we use parallel, untied generators to form the parameters of each layer. In order for that to work, we observe that parameters between network layers are not independent, but are strongly correlated. The parameters of each layer depend on the output and the parameters of the preceding layers. Therefore, the generated parameters must be correlated to produce well-performing neural networks. We propose to add a *Mixer* Q which maps $s \sim \mathcal{S}$ to the

mixed latent space $Z \in \mathbb{R}^{Nd}$, $d < j$. An Nd-dimensional $Q(s)$ is then partitioned

into N layer embeddings $[q_1, \ldots, q_N]$, each being a d-dimensional vector. Finally, we use N parallel generators $G = \{G_1(q_1) \ldots G_N(q_n)\}$ to generate the parameters θ_G for all layers in $\mathcal{F}$.

After generation, we can evaluate the new model $\mathcal{F}(x, \theta_G)$ on the training set. We define an objective which minimizes the error of generated parameters with respect to a task loss $\mathcal{L}$:

$$\inf_{G,Q} \mathbb{E}_{s\sim\mathcal{S}} \mathbb{E}_{(x,y)\sim(X,Y)} \left[\mathcal{L}(\mathcal{F}(x; G(Q(s))), y)\right] \tag{2.1}$$

At each training step we generate a different network $G(Q(s))$ from a random $s \sim S$, and then evaluate the loss function on a mini-batch from the training set. The resulting loss is backpropagated through the generators until θ_G minimizes the target loss $\mathcal{L}$.

The formulation in (2.1) presents a problem: the codes sampled from $Q(s)$ will certainly collapse to the maximum likelihood (ML) estimate (when $\mathcal{L}$ is a log-likelihood). This means that the generators will learn a very narrow approximation of Θ, and indeed we see this happen in tables **??** and **??**. To assure that the parameters are well distributed, we added an adversarial constraint on the mixed latent space $\mathcal{D}(Q(s))$ so that it should not deviate too much from a Gaussian prior $\mathcal{P}$. This constraint makes it closer to the generated parameters and ensures that $Q(s)$ itself does not collapse to always outputting the same latent code. With this we arrive at the HyperGAN objective:

$$\inf_{G,Q} \mathbb{E}_{s\sim\mathcal{S}} \mathbb{E}_{(x,y)\sim(X,Y)} \left[\mathcal{L}(\mathcal{F}(x; G(Q(s))), y)\right] - \beta\mathcal{D}(Q(s), \mathcal{P}) \tag{2.2}$$

Where β is a hyperparameter, and $\mathcal{D}$ is the regularization term which penalizes the distance between the prior and the distribution of latent codes. In practice $\mathcal{D}$ could be any distance function between two distributions. We choose to parameterize $\mathcal{D}$ as a discriminator network D that output probabilities, and use the adversarial loss [17] to approximate $\mathcal{D}(Q(s), \mathcal{P})$. Note that while $\mathcal{P}$ and $\mathcal{S}$ are both multivariate Gaussians, they are distinct distributions as seen in figure 2.1.

$$\mathcal{D} := -\sum_{i=1}^{N} \left(\log D(p_i) + \log(1 - D(q_i))\right) \tag{2.3}$$

Note that we find it difficult to learn a discriminator in the output (parameter) space because the dimensionality is high and there is no structure in those parameters to be uti-

lized as in images (where CNNs can be trained). Our experiments show that regularizing in the latent space works well. We hypothesize this is because of overparametrization in θ. The latent space initializes with random projections, which have a restricted isometry property according to the Johnson-Lindenstrauss lemma, hence if the mixed latent factor q is Gaussian, a random projection from it preserves the distance and diversity from q to θ. If θ is indeed severely overparametrized, and it is possible to generate diverse parameters by maximizing the likelihood, then the generators would not collapse to a single θ, since that would require breaking the restricted isometry from the initialization.

This framework is general and can be adapted to a variety of tasks and losses. In this work, we show that HyperGAN can operate in both classification and regression settings. For multi-class classification, the generators and mixer are trained with the cross entropy loss function:

$$\mathcal{L}_H = M^{-1} \sum_{i=1}^{M} y_i \log\left(\mathcal{F}(x_i;\theta)\right) \quad \text{where} \quad \theta = \{G_1(q_1), \ldots, G_n(q_n)\} \tag{2.4}$$

For regression tasks we replace the cross entropy term with the mean squared error function:

$$\mathcal{L}_{mse} = M^{-1} \sum_{i=1}^{M} (y_i - \mathcal{F}(x_i;\theta))^2 \quad \text{where} \quad \theta = \{G_1(q_1), \ldots, G_n(q_n)\} \tag{2.5}$$

2.3.1 Learning without an Explicit Target Distribution

In implicit generative models such as GAN or WAE [48], it is always necessary to have a collection of inlier data points to train with. Inlier samples come from the distribution that is being estimated, and learning without them is difficult. HyperGAN does not have a given set of inlier points to train with. Instead, HyperGAN uses the error on the target loss to estimate the distance between the generated samples $\theta_G \in \Theta_G$ and the target samples $\theta \in \Theta$. We note that θ_G represents the maximum likelihood estimate of $\mathcal{F}(x;\theta)$. (2.6) shows that by minimizing the error of the ML estimate on the log-likelihood, we are in fact minimizing the KL divergence between the target distribution Θ and the generated samples $\theta_G \in \Theta_G$.

$$\begin{aligned}\inf_{\theta_G} D_{KL}(P(x|\theta)||P(x|\theta_G)) &= \inf_{\theta_G} \mathbb{E}_{P(x|\theta_G)}\left[\log P(x|\theta) - \log P(x|\theta_G)\right] \\ &= \inf_{\theta_G} \mathbb{E}_{P(x|\theta_G)}\left[-\log P(x|\theta_G)\right] \end{aligned} \tag{2.6}$$

Where the term $\log P(x|\theta)$ is the entropy of the target distribution and does not contribute to the optimization problem. The derived likelihood function is flexible and used widely across different domains. In our work it is represented by the cross entropy and MSE losses that we study.

Its useful to note here how HyperGAN is distinct from other density estimators such as GAN and WAE. These methods use a likelihood function, such as MSE, to estimate the quality of the samples (in GANs the likelihood function is learned via the discriminator). The likelihood is given as the distance from generated samples, to given samples of the target distribution. HyperGAN instead estimates both the target and the approximation. We assume the target distribution exists, and update our approximation Θ_G to better match Θ. If we have samples of the target distribution, we can reduce HyperGAN to a GAN (by moving the discriminator to the output space). We thoroughly explore the connections and differences to GANs and WAE in **??**. HyperGAN is different from fully probabilistic approaches such as MNF [30] since one cannot compute the probability of the generated θ, nor can one encode a generated θ back to the latent spaces.

2.4 Experiments

2.4.1 High Level Description and Experimental Setup

We conduct a variety of experiments to show HyperGAN's ability to achieve both high accuracy and obtain accurate uncertainty estimates. First we show classification performance on both MNIST and CIFAR-10 datasets. Next we examine HyperGAN's ability to learn the variance of a simple 1D dataset. We perform experiments on anomaly detection: testing HyperGAN on notMNIST, and 5 classes of CIFAR-10 which are hidden during training. We also examine adversarial examples as extreme cases of off-manifold data, and test our robustness to them. In our experiments we compare with [30] (MNF) as well as standard ensembles.

In all experiments we report results with two HyperGANs, one trained on MNIST and another on CIFAR-10. Both of our models take a 256 dimensional sample of $\mathcal{S}$ as input, but have different sized mixed latent spaces. The HyperGAN for the MNIST experiments consists of three weight generators, each using a 128 dimensional latent point as input. The target network for the MNIST experiments is a small two layer convolutional network, using leaky ReLU activations and 2x2 max pooling after each convolutional layer. Our HyperGAN trained on CIFAR-10 used 5 weight generators and latent points with dimensonality 256. The target architecture for CIFAR-10 tests consists of three convolutional layers, each followed by leaky ReLU and 2x2 max pooling. The exact architectures we used for the target networks is given in tables 2.1 and 2.2. In our experiments we used the same network architecture across different methods. Note that our architecture is different from the LeNet-5 used in MNF, yet we see similar results as reported in [30].

Table 2.1: MNIST HyperGAN Target Size

Layer	Latent size	Output Layer Size
Conv 1	128 x 1	32 x 1 x 5 x 5
Conv 2	128 x 1	32 x 32 x 5 x 5
Linear	128 x 1	512 x 10

Table 2.2: CIFAR-10 HyperGAN Target Size

Layer	Latent Size	Output Layer Size
Conv 1	256 x 1	16 x 3 x 3 x 3
Conv 2	256 x 1	32 x 16 x 3 x 3
Conv 3	256 x 1	32 x 64 x 3 x 3
Linear 1	256 x 1	256 x 128
Linear 2	256 x 1	128 x 10

HyperGAN Details

For our HyperGAN network architectures we use 2 layer MLPs with 512 units each and exponential rectifier activations [8] for the encoder, weight generators, and discriminator. We found in a pilot study that larger networks in fact offered little performance benefit, and ultimately hurt scalability. In all experiments, we pretrain the encoder so that the mean and covariance of Q_z match $\mathcal{P}_z$. It should be noted that HyperGAN is flexible with respect to the exact architecture. The number of layers or the nonlinearity may be varied without harming HyperGANs ability to model the target distribution. We trained our HyperGAN on MNIST using less than 1.5GB of memory on a single GPU, while CIFAR-10 used just 4GB, making HyperGAN surprisingly scalable.

In Table 2.4 we show some statistics of the networks generated by HyperGAN on MNIST. We note that HyperGAN can generate very diverse networks, as the variance of network parameters generated by the HyperGAN is significantly higher than standard training from different random initializations.

HyperGAN - MNIST			
	Conv1	Conv2	Linear
Mean	7.49	51.10	22.01
σ^2	1.59	10.62	6.01
Standard Training - MNIST			
	Conv1	Conv2	Linear
Mean	27.05	160.51	5.97
σ^2	0.31	0.51	0.06

Table 2.3: 2-norm statistics on the layers of a population of networks sampled from HyperGAN, compared to 10 standard networks trained from random initializations.

2.4.2 Classification

First we evaluate the classification accuracy of HyperGAN on MNIST and CIFAR-10. Classification serves as an entrance exam into our other experiments, as the distribution we want to learn is over parameters which can effectively solve the classification task.

HyperGAN - CIFAR-10					
	Conv1	Conv2	Conv3	Linear1	Linear2
Mean	7.65	15.77	41.86	16.36	5.85
σ^2	4.13	38.52	104.26	13.22	0.91
Standard Training - CIFAR-10					
	Conv1	Conv2	Conv3	Linear1	Linear2
Mean	5.13	15.19	16.15	11.79	2.45
σ^2	1.19	4.40	4.28	2.80	0.13

Table 2.4: 2-norm statistics on the layers of a population of CIFAR-10 networks sampled from HyperGAN, compared to 10 standard networks trained from different random initializations.

We test with both single network samples, and ensembles. For our ensembles we average predictions from N sampled models with the scoring rule $p(y|x) = \frac{1}{N}\sum_{n=1}^{N} p_n(y \mid x, \theta_n)$. Our target network for the MNIST experiments is a small two layer convolutional network, using leaky ReLU activations and 2x2 max pooling after each convolutional layer. Our target architecture for CIFAR-10 tests consists of three convolutional layers, each followed by leaky ReLU and 2x2 max pooling. The sizes of each layer can be found in tables 2.2 and 2.1. It should be noted that we did not perform fine tuning, or any additional training on the sampled networks. The results are shown in Table 2.5. We generate ensembles of different sizes and compare against both Bayesian [30] [26] and non-Bayesian [27] methods, as well as MC dropout [16]. We outperform all other methods by using a 100 network ensemble, across all datasets. We test on the datasets MNIST and CIFAR-10. Within these datasets we explore additional tests. MNIST and CIFAR 5000 involve training only on the first 5000 samples of the training set. CIFAR-5 only uses the first 5 classes for training and testing, so the accuracy here is noticeably greater.

2.4.3 1-D Toy Regression Task

We next evaluate the ability of HyperGAN to fit a simple 1D function from noisy samples. This dataset was first proposed by [22], and consists of a training set of 20 points drawn uniformly from the interval $[-4, 4]$. The targets are given by $y = x^3 + \epsilon$ where $\epsilon \sim$

Method	MNIST	MNIST 5000	CIFAR-5	CIFAR-10	CIFAR-10 5000
1 network	98.64	96.69	84.50	76.32	76.31
5 networks	98.75	97.24	85.51	76.84	76.41
10 networks	99.22	97.33	85.54	77.52	77.12
100 networks	**99.31**	**97.71**	**85.81**	**77.71**	**77.38**
Deep Ensembles	99.30		79.00		
MNFG	99.30		84.00		
BHN	98.63	96.51		74.90	
MC Dropout	98.73	95.58	84.00	72.75	

Table 2.5: Classification performance of HyperGAN on MNIST and CIFAR-10.

$\mathcal{N}(0, 3^2)$. We used the same target architecture as in [22] [27] and [30]: a one layer neural network with 100 hidden units and ReLU nonlinearity. For HyperGAN we use two layer generators, and 128 hidden units across all networks. Because this is a small task, we use only a 64 dimensional latent space. MSE loss is used as our target loss function to train HyperGAN.

Results in figure 2.2 show the results of HyperGAN-generated ensembles with 5, 10, and 100 generated networks respectively. HyperGAN clearly learns the target function and captures the variation in the data well. In addition, it can be seen that sampling more networks to compose a larger ensemble improves predictive uncertainty as we sample farther from the mean of the training data.

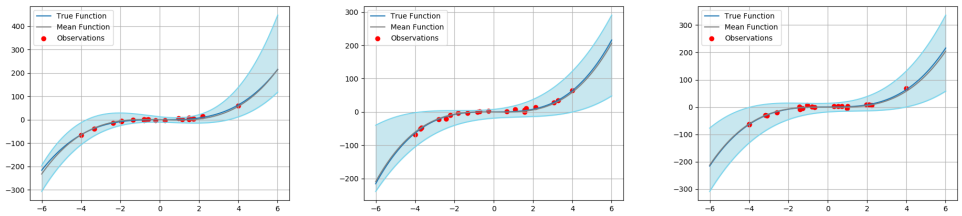


Figure 2.2: Results of HyperGAN on the 1D regression task.

2.4.4 Anomaly Detection

To test our uncertainty measurements, we perform the same experiments as [30], [27]; we measure the total entropy in predictions from HyperGAN-generated networks. For MNIST experiments we train a HyperGAN on the MNIST dataset, and test on out-of-distribution notMNIST, which consists of 28x28 binary images of letters. In this setting, we want the softmax probabilities on inlier MNIST examples to have maximum entropy - a single large activation close to 1. On off-manifold data we want to have equal probability across predictions. We test our CIFAR-10 model by just training on the first 5 classes, and we use the latter 5 classes as out of distribution examples. To build an estimate of the predictive entropy we sample multiple networks from HyperGAN per example, and measure their predictive entropy.

In Fig. 2.3 we show that HyperGAN can separate CIFAR-10 inlier and outlier samples much better than MNF or standard ensembles. HyperGAN is less certain about data it does not recognize, as the probability of a low entropy prediction is overall lower on outliers. On notMNIST we also show separation, though HyperGAN is also overall less confidant about inliers then MNF. Conventionally trained ensembles without the HyperGAN, referred to as L2 networks in the figure, are highly overconfident on outliers and cannot provide a notion of uncertainty.

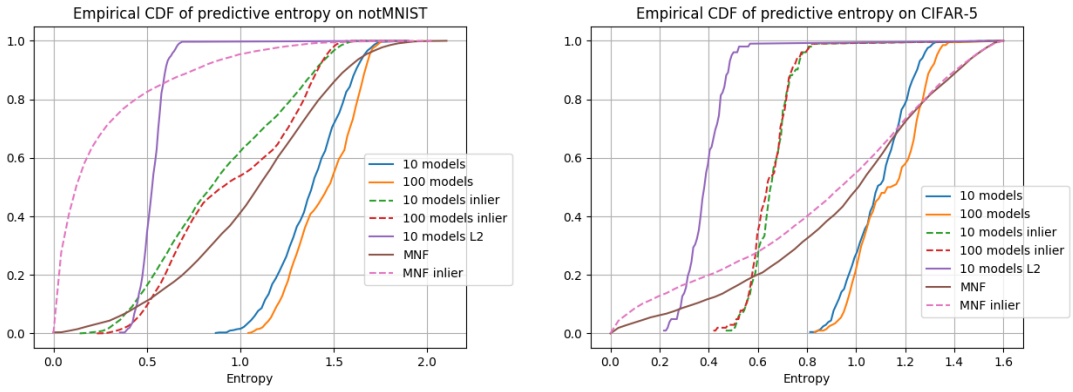


Figure 2.3: Empirical CDF of the predictive entropy on out of distribution datasets notMNIST, and 5 classes of CIFAR-10 unseen during training. Solid lines denote tests on the respective out of distribution data, while the dashed lines denote entropy on inlier examples (MNIST and CIFAR-10). L2 refers to conventional ensembles trained separately without a HyperGAN

2.4.5 Adversarial Detection

We employ the same experimental setup to the detection of adversarial examples, an extreme sort of off-manifold data. Adversarial examples are often optimized to lie within a small neighborhood of a classifier's decision boundaries. They are created by adding perturbations in the direction of the greatest loss with respect to the model's parameters. Because HyperGAN learns a distribution over parameters, it should be more robust to attacks. We generate adversarial examples using the Fast Gradient Sign method (FGSM) [18] and Projected Gradient Descent (PGD) [32]. FGSM adds a small perturbation ϵ to the target image in the direction of greatest loss. FGSM is known to underfit to the target model, hence it may transfer well across many similar models. In contrast, PGD takes many steps in the direction of greatest loss, producing a stronger adversarial example, at the risk of overfitting to a single set of parameters. This poses the following challenge: to detect attacks by FGSM and PGD, HyperGAN will need to generate diverse parameters to avoid both attacks.

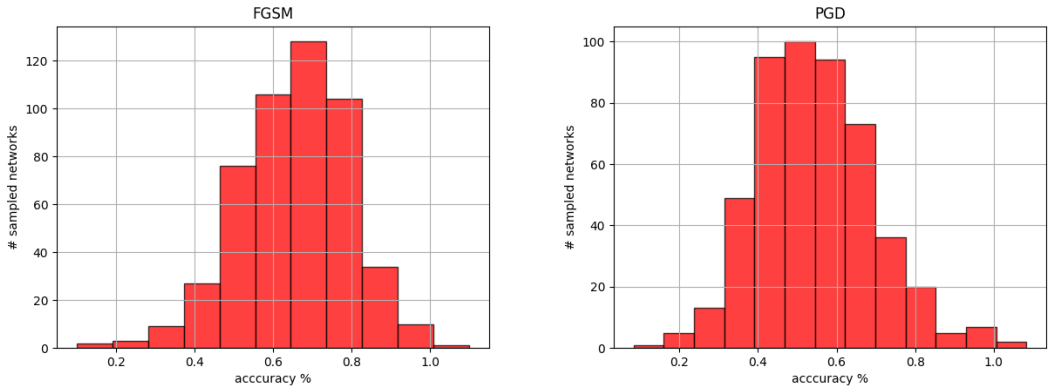


Figure 2.4: Diversity of predictions on adversarial examples.

To detect adversarial examples, we first hypothesize that a single adversarial example will not fool the entire space of parameters learned by HyperGAN. If we then evaluate adversarial examples against many generated networks, then we should see a high level of disagreement among predictions for any individual class. In this case, we define disagreement as a function of entropy in the softmax probabilities at the output of the network. Given the unnormalized outputs x of N models, we compute the disagreement

d across K classes:

$$d = -\sum_i p_i \log p_i \quad \text{where} \quad p_i = N^{-1} \sum_{i=1}^{N} \frac{\exp f(x)_i}{\sum_{k=1}^{K} \exp f(x)_k}$$

where $f(x)_i$ refers to the logits of class i.

Adversarial examples have been shown to successfully fool ensembles [12], but with HyperGAN one can always generate significantly more models that can be added to the ensemble for the cost of one forward pass, making it hard to attack against. In Fig. 2.4 we test HyperGAN against adversarial examples generated to fool one network. It is shown that while those examples can fool $50\% - 70\%$ of the networks generated by HyperGAN, adversarial examples do not fool all generated networks.

We compare the performance of HyperGAN with ensembles of $N \in \{5, 10\}$ models trained on MNIST with normal supervised training. We fuse their logits (unnormalized log probabilities) together as $l(x) = \sum_{n=1}^{N} w_n l_n(x)$ where w_n is the nth model weighting, and l_n is the logits of the nth model. In all experiments we consider uniformly weighted ensembles. For HyperGAN we simply sample from parameter space to create as many models as we need, and similarly fuse their logits together. Specifically we test ensembles with $N \in \{5, 10, 100, 1000\}$ members each. Here adversarial examples are generated by attacking the ensemble directly. For HyperGAN, we attack an ensemble of networks, but test with a new ensemble of equal size.

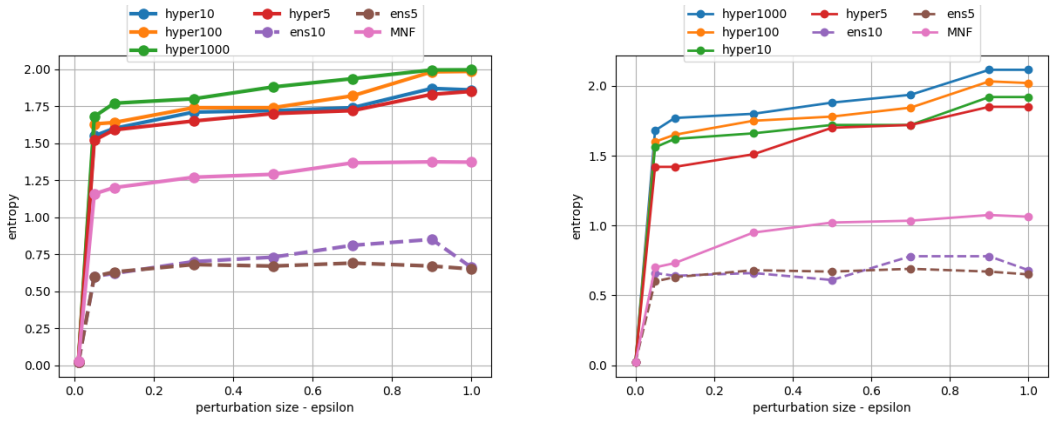


Figure 2.5: Entropy of predictions on FGSM and PGD adversarial examples.

For the purposes of detection, we compute the entropy within the predictive distribution of each of the ensemble members to score the example on the likelihood that it was drawn from the training distribution. Figure 2.5 shows that HyperGAN easily identifies adversarial examples as being out-of-distribution. HyperGAN is especially suited to this task as adversarial examples are optimized against parameters - parameters which HyperGAN can change. We find that we can successfully detect over 97% of adversarial examples, with a low false positive rate for both attacks just by thresholding the entropy.

2.5 Discussion and Future Directions

We have proposed a generative, solution to parameter selection which performs strongly on detecting out-of-distribution samples, as well as classification. Training a GAN to learn a probability distribution over parameters allows us to non-deterministically sample diverse, performant networks which we can use to form ensembles that can give good uncertainty estimates. Our method is ultimately scalable to any number of networks in the predicting ensemble, requiring just one forward pass to generate a new set of parameters and a low GPU memory footprint. We showed that we can generate models with significant variation over the learned distribution and thus provide uncertainty estimates on outlier data. Our HyperGAN can be readily extended to generate parameters for a variety of architectures such as MLPs, CNNs, etc. We hope that this will encourage the community to consider other generative approaches to learning the manifold of neural networks. There is still much room for exploration, we believe that learning a low dimensional manifold of performant neural networks could be useful for a variety of domains including meta learning and reinforcement learning. In the future we wish to explore agent curiosity and exploration policies aided by uncertainty measurements from HyperGAN, or explore transfer learning by learning a manifold of neural networks which can solve more than one task.

Chapter 3: Robust Neural Networks

In addition to detecting out of distribution examples, we acknowledge that detection may not be a strong enough result. In situations where a prediction must be made, abstinence can also be a fatal action. In the case of adversarial examples, there is inlier data hidden beneath some perturbation. We want neural networks to be robust to inputs which actively seek to fool the predictor. In this chapter we explore robustness to adversarial examples in particlar, restricting ourselves from the larger uncertainty discussion in chapter 2. For this purpose we revisit in greater detail, the theory of adversarial examples.

Recent analysis of deep neural networks has revealed their vulnerability to carefully structured adversarial examples. Many effective algorithms exist to craft these adversarial examples, but performant defenses seem to be far away. In this work, we explore the use of edge-aware bilateral filtering as a projection back to the space of natural images. We show that bilateral filtering is an effective defense in multiple attack settings, where the strength of the adversary gradually increases. In the case of an adversary who has no knowledge of the defense, bilateral filtering can remove more than 90% of adversarial examples from a variety of different attacks. To evaluate against an adversary with complete knowledge of our defense, we adapt the bilateral filter as a trainable layer in a neural network and show that adding this layer makes ImageNet classifiers images significantly more robust to attacks. When trained under a framework of adversarial training, we show that the resulting model is hard to fool with even the best attack methods.

3.1 Adversarial Examples

Deep neural networks are known to be vulnerable to targeted perturbations added to benign inputs. The perturbed inputs, known as *adversarial examples*, can cause a classifier to output highly confident, but incorrect predictions. The majority of prior work has studied adversarial examples in the context of computer vision, where they pose the clearest threat. Small perturbations, imperceptible to humans, can be added to input

images that cause a classifier to output false predictions. Because of the particular success of neural networks in computer vision, these models are being deployed in areas such as autonomous driving, facial recognition, and malware detection. Recent work has shown that these systems are vulnerable in the real world to adversarial examples [14], which makes the problem of resisting adversarial attacks a growing concern.

There have emerged two central lines of research for defending against adversarial examples. *Denoising* approaches attempt to remove the adversarial perturbations from the inputs as a preprocessing step. This is often done by filtering, or by projecting the input to a lower dimensional space that cannot represent high frequency perturbations [43] [45]. These methods often lead to high accuracy, even on difficult datasets like ImageNet. But it has been shown that an attacker with knowledge of the defense can successfully circumvent them [5]. On the other hand, *Adversarial training* methods use principles from robust optimization to train models which resist adversarial attacks. Under the adversarial training framework, adversarial examples are combined with the natural training set to increase the model's robustness to attacks. These methods are expensive, requiring many more training examples, and have not been shown to scale well to natural image datasets such as ImageNet.

This paper explores the utility of bilateral filtering as both a denoising defense and a useful addition to adversarial training. Bilateral filtering is a classic approach in computer vision for edge-aware smoothing. Because natural images are more likely piecewise-smooth while adversarial perturbations are less likely to be, we hypothesize that bilateral filtering would be able to filter out adversarial noises. Indeed, in experiments we found that with appropriate parameters, a plain bilateral filter can recover **99%** of the adversarial images so that a classifier can predict the original label.

Furthermore, we introduce BFNet: an end-to-end model incorporating bilateral filtering as a differentiable layer. With BFNet, it is possible to examine the performance of white-box attacks trying to bypass our bilateral filtering defense. We show that BFNet is naturally robust to attacks from many such adversaries, greatly reducing the strength of both L_∞ and L_2 attacks on the ImageNet dataset.

Finally, we combine bilateral filtering with adversarial training, and achieve state-of-the-art results on MNIST and CIFAR10. Our method works with zero knowledge of either the network or any incoming attack, making it applicable to a variety of models and datasets. We demonstrate the versatility of our approach by testing bilateral filter

based defenses in the strongest adversary with full access to our defense and can optimize against it, while the defense having no knowledge

1. Against adversaries with no knowledge of our defense, we show that a vanilla bilateral filter with fixed parameters can reliably remove adversarial perturbations created by a variety of strong adversaries.
2. We further develop an adaptive filtering network which predicts bilateral filter parameters from an input image, to combat an adversary who employs a larger attack repertoire. We test our network on the ImageNet dataset, using Inception V3 and InceptionResNet V2 classifiers. We found that our adaptive filtering network successfully removes adversarial perturbations from natural images, recovering the clean class label.
3. The strongest adversary has access to our defense, and can optimize against it. We show that BFNet is naturally robust to such perfect-knowledge attacks on ImageNet. We also combine BFNet with adversarial training to further increase its resistance to attack. In particular, we evaluated the ability of adversaries to attack our network on the MNIST and CIFAR10 datasets, and achieve state of the art results for PGD and FGSM attacks.

3.2 Current Attacks and Defenses

3.2.1 Adversarial Attacks

There have been many proposed attacks for creating adversarial examples. We give a brief description of the six attacks that we used to test our models.

A. Projected Gradient Descent (PGD)

When measuring the robustness of a model to adversarial attack. It is helpful to limit the strength of the adversary by generating perturbations within a bounded magnitude ϵ. In [32], generating an adversarial example is the task of solving the objective:

$$\max_{\delta \leq \epsilon} L(\theta, x + \delta, y_{true})$$

PGD is used to minimize this objective under a loss function L, yielding an image with perturbations with magnitude less than ϵ with respect to the max norm, and achieves the highest possible loss.

B. Fast Gradient Sign Method (FGSM)

FGSM [18] is a one step linearization of the above objective. FGSM finds adversarial examples by assuming linearity at the decision boundary. Given an image x, we find a perturbation η under the max norm:

$$\eta = \epsilon \cdot sign(\nabla_x L(\theta, x, y))$$

Where θ is the parameters of the network, y is the original label, and L is the loss function used to train the network.

C. Momentum Iterative Method

The Momentum Iterative Fast Sign Gradient Method (MI-FGSM) [12] is an iterative version of the FGSM attack. MI-FGSM moves pixel values linearly along the gradient toward the decision boundary. MI-FGSM improves on FGSM by introducing a momentum term into gradient calculation.

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_x L(x_t^*, y)}{\|\nabla_x L(x_t^*, y)\|_1}$$

The gradient is then used to iteratively update the image

$$x_{t+1}^* = x_t^* + \alpha \cdot (g_{t+1})$$

The authors claim that simply using an iterative FGSM leads to greedy overfitting of the decision boundary, and thus falls into local poor maxima. Adding momentum stabilizes the update direction and creates a stronger adversarial example.

D. L-BFGS-B

Szegedy, et al. [47] used box-constrained L-BFGS to generate adversarial examples with minimal distortion under the L_2 norm. Given a natural image x and a target class y_{true}, the adversarial objective is as follows

$$\min \Big[c \cdot ||x - (x + \delta)||_2^2 + L(x + \delta, y_{target}) \Big]$$

Where δ is the adversarial perturbation, L is the loss function, and the parameter c controls the trade-off between the magnitude and strength of the perturbation.

E. Carlini & Wagner Attack (L_2)

Carlini, Wagner, [7] proposed three iterative attacks which create adversarial examples under the L_0, L_2, and L_∞ norms. In this work we consider the most powerful attack, the white-box L_2 attack. Specifically, they minimize

$$\min ||\frac{1}{2}(\tanh(w) + 1)x||_2^2 + cf(\frac{1}{2}(\tanh(w) + 1))$$

$$\text{Where } f(x') = \max(\max\{Z_i(x') : i \neq t\} Z_t(x'), -\kappa)$$

Here, t is the target label, Z refers to the logits of the network, κ controls the confidence of the new classification, and the $\frac{1}{2}\tanh$ term constrains the result to pixel space.

F. DeepFool

Deepfool is an iterative, first order method used to find minimal distortion under the L_2 norm [36]. Deepfool linearizes the classifier itself and performs gradient descent until the image is misclassified. The DeepFool objective is as follows:

$$\min_{\delta} \|\delta\|_2 \quad \text{subject to} \quad argmax\, f(x) \neq argmax\, f(x + \delta)$$

In addition to the attacks listed above, other methods have been proposed. L_0 attacks such as [39] choose to measure adversarial perturbations by the minimum change necessary to produce an incorrect prediction.

3.2.2 Adversarial Defenses

There is a growing body of work on defenses against adversarial attacks [38],[40], [29], [51]. An averaging filter was studied in [28]. JPEG compression was studied in [13] [10], and was found to be effective at removing adversarial perturbations. However, JPEG encoding is not differentiable, hence its performance when the adversary has knowledge of the defense is unknown. Our bilateral filtering approach is fully differentiable hence we can test it against counter-attacks.

Other recent defenses attempt to remove adversarial perturbations by projecting inputs back onto the real data manifold [34]. One method [45] projects inputs using a generative adversarial network. Given a normal or adversarial image, a generator is trained to produce a image from the normal data distribution. This method also did not test against counter-attacks, and has been shown to be successfully fooled by the CW attack [7]. Our approach can also be seen as a projection back to the data manifold, where we impose the constraint that the resulting image must be piecewise-smooth. By fixing the filter approach, we would likely not overfit significantly to the training set and remain effective under counter-attacks.

On the other hand, adversarial training methods [19], [32], [44], [49] combine adversarial examples with the natural training set to increase the robustness of the model to adversarial attacks. These approaches are promising as they attempt to provide a guarantee on both the type of adversary and the magnitude of the perturbation they are resistant to. In practice however, these methods are hard to scale as they require expensive computation in the inner training loop to generate adversarial examples. When training on a large dataset such as ImageNet, generating a sufficient amount of strong adversarial examples can be intractable. This problem has been mitigated by training against a weak adversary like FGSM [49] which can quickly generate adversarial examples. But training models that are robust to strong adversaries on ImageNet or CIFAR-10 is still an open problem.

3.3 Bilateral Filtering

In this section we present our motivation for using a bilateral filter as the building block for our robust defenses. We give a quick discussion of the theory of edge aware filtering and how it applies to adversarial examples. We show that it can remove adversarial perturbations crafted by many strong attacks. We then describe our end to end approach for training a classifier with a bilateral filter as a differentiable filtering layer. Finally, we recognize the current state of the art method for training models robust to adversarial attacks, adversarial training, and we show that our method can extend adversarial training to create still more robust models. We will first show the utility of bilateral filter against simple attacks without knowledge of the network, then introduce BFNet with bilateral filtering as a differentiable layer, so that we can evaluate attacks with knowledge of our defense.

3.3.1 Threat Model

Prior work often presents adversarial attacks and defenses as being either a white-box or black-box attack. In the white-box threat model, the attacker has full access to training data, model parameters and architecture. The black-box threat model considers the case where the attacker has little to no knowledge about the model architecture, parameters, or training data. In the most restrictive setting, the attack may only have access to the argmax output of a softmax operation. In this work we consider white-box attacks, as they are categorically stronger adversaries than black-box attacks. We consider three different levels of robustness for our bilateral filter defenses. We measure robustness against an attacker without knowledge of the defense, a changing attacker who may employ many different adversarial attacks, and finally an adversary who can mount a perfect-knowledge attack.

3.3.2 Recovering Adversarial Images with a Bilateral Filter

The bilateral filter is a non-linear Gaussian filter that is commonly used to smooth image gradients while preserving sharp edges. For an image I, window Ω centered at pixel p,

the bilateral filter is formulated as a domain function G_s, and a range function G_r:

$$I_{filtered}(p) = \frac{1}{W_p} \sum_{q \in \Omega} G_s(||\mathbf{p} - \mathbf{q}||) G_r(\|I_p - I_q\|)\, I_q$$

where the normalization term W_p is:

$$W_p = \sum_{q \in \Omega} G_s(\|\mathbf{p} - \mathbf{q}\|) G_r(\|I_p - I_q\|),$$

$G_s(x) = \exp(-\frac{x^2}{2\sigma_s^2})$ and $G_r(x) = \exp(-\frac{x^2}{2\sigma_r^2})$ are Gaussian filters, and σ_s and σ_r are parameters which control the strength of the domain and range functions respectively. Each neighboring pixel is assigned a weight according to both spatial closeness and value difference. Hence, if the color of the pixels p and q are very different, then q will affect the filtered image at pixel p very little. At sharp image boundaries, this would effectively lead to smoothing on only one side of the boundary, since the other side would have very different color. Hence, sharp boundaries can be preserved and oversmoothing or blurring that are commonly seen in Gaussian smoothing or averaging can be prevented. In Fig.3.1 one can see the effect of denoising an adversarial example created with L-BFGS-B, where an averaging filter will leave the image significantly blurred, but bilateral filtering would preserve the edges. More images are shown in the appendix in Fig. B.1.

To test the efficacy of the bilateral filter to recover clean inputs from adversarial examples, we generated a set of adversarial examples from a range of powerful adversaries. Our first approach was to manually tune parameters for each input image, to test the effective range of parameters which could recover the original label from an adversarial example. We found that with carefully chosen parameters, the corrupted labels could indeed be recovered. Our experiments showed that the small perturbations created by iterative methods like the Carlini & Wagner attack and DeepFool were easier to remove with a bilateral filter than the larger perturbations created with one step attacks. To remove perturbations generated by iterative attacks, we used small kernels 3 - 5 pixels wide, and σ_s, σ_r values of 0.5. Filtering with larger kernel sizes offers no benefit, as the resulting images from iterative attacks have imperceptible perturbations which are removed with small filters. One step attacks perturb every pixel in the image with the same magnitude of noise. As a result, we increased kernel width to 7 and σ_s to 3, hold-

ing σ_r constant. These parameters reliably removed adversarial perturbations from L_∞ attacks with a bounded distance of 0.3, as well as unbounded L_2 attacks. The results can be found in Table 3.1.

Figure 3.1: (a) The original LBFGS-B adversarial image, (b) The image after 3x3 bilateral filtering and (c) The image after 3x3 averaging filtering. The bilateral filter is superior since it removes small perturbations while preserving sharp edges in the image, keeping it from becoming blurry

Network	FGSM	MI-FGSM	DeepFool	CW (L_2)	L-BFGS
Inception V3	97.0	97.5	98.8	99.2	97.8
InceptionResNet V2	94.2	98.4	96.3	98.8	95.1
ResNet V2	96.5	98.0	96.1	98.1	98.0

Table 3.1: Recovery performance for manually chosen bilateral filter parameters.

3.3.3 Adaptive Filtering

One caveat to the above approach, is that the parameters for the bilateral filter must be carefully chosen to be able to recover the accuracy and confidence of the original classification. Large values for the parameters σ_s and σ_r can create an excessively blurred image, and a small filter size K may capture insufficient information to remove the adversarial perturbations. With this in mind, we train a small network which will predict the parameters of the bilateral filter (K, σ_s, σ_r) for an input image. This network will serve as a cheap preprocessing step that will remove adversarial perturbations without

affecting the underlying class label.

To build our classifier we first extract information about the distribution of pixel gradients by convolving the input with a Sobel filter in the x and y direction. Because adversarial attacks directly change values of the input, adversarial examples will often have larger gradients in the x and y direction than natural images. We concatenate the gradient map depth-wise with the input image, and use three dilated convolutional layers with 64, 128, and 256 filters respectively, followed by 2x2 max pooling and a linear layer of 64 units. We use a dilation rate of 2 for each convolutional layer. Note this experiment is stand-alone and it is not utilized in the BFNet proposed in the next section.

3.3.4 BFNet: Bilateral Filter as A Trainable Layer

The main idea of BFNet is to always preprocess the input image with bilateral filter before inputting it into the CNN. Namely, instead of computing $f(x)$ where f is learned by a deep network, always computing $f(BF(x))$ instead. Hence we can then optimize for attacks that have full knowledge and gradients about our defense. This has two utilities, one is to examine the robustness of the defense, and secondly we can add the newly generated adversarial examples back to the training set of the network, in order to perform adversarial training.

A brute-force implementation of the bilateral filter has a $\mathcal{O}(n^2)$ cost associated with computing the response of individual pixels, making it the most expensive operation in the graph. To reduce computation time, We choose as our preprocessing function the Permutohedral Lattice implementation of the bilateral filter [1], which is also fully differentiable and can be computed in $\mathcal{O}(n)$ time. This can then be attached as the first layer to any other network, and the bilateral filter parameters can be trained jointly with other parts of the network.

3.3.5 Adversarial Training

It has been shown that under the white-box threat model, using a denoiser as the only defense is insufficient to stop the strongest adversarial attacks. Currently the most promising direction for training models robust to adversarial attacks is adversarial training. Despite continuing progress on both MNIST and CIFAR10, adversarial training is

still very expensive, and performs worse than denoising approaches on the same datasets. We propose an approach combining adversarial training with BFNet, giving a robust, performant classifier on different threat models.

Following [32],[5], [7], the adversarial training framework can be expressed as the following saddle point problem with model parameters θ, and input x with true label y:

$$\min_{\theta} f(x;\theta) \quad \text{where} \quad f(x;\theta) = \mathbb{E}_{(x,y)\sim D}\Big[\max_{\delta} L(\theta, x+\delta, y)\Big],$$

where a solution to the inner maximization problem represents the *most* adversarial example within some perturbation budget. Solving the outer minimization problem yields a classifier which is robust to the above adversary. [32] showed that PGD could reliably solve the inner maximization problem without linearization, and is thus a better adversary to train against than FGSM.

We propose a modification to the above saddle point formulation which incorporates the BFNet:

$$\min_{\theta} f(BF(x)) \quad \text{where} \quad f(x) = \mathbb{E}_{(x,y)\sim D}\Big[\max_{\delta} L(\theta, x+\delta, y_{true})\Big]$$

where $BF(x)$ is the bilateral filter in BFNet.

3.4 Experiments

3.4.1 Adaptive Filtering Model

In this section we show that our adaptive filtering model can correctly predict filtering parameters which will restore an adversarial input. To test this, we generate a dataset of 1,000 adversarial images from the ILSVRC 2012 validation set with five different attacks: Projected Gradient Descent with 40 steps (PGD), Box constrained L-BFGS, The Carlini & Wagner L_2 attack (CW), The Momentum Iterative FGSM (MIM), FGSM, and DeepFool. Where applicable, we constrain the perturbations to an ϵ-ball of radius 0.3 from the training example. Source images have been normalized to a range of $[-1, 1]$.

To construct our training set we use a separate 1,000 images generated from each of the attacks in table 1. For each image, we collect labels in the form of triples (K, σ_s, σ_r), K denotes the kernel size, and σ_s, σ_r are the standard deviation for the spatial and range

kernels respectively. Given any adversarial example, there may be many permutations of parameters for the bilateral filter that successfully denoise the input. For this reason we collect a maximum of 10 different parameter configurations for each image in our training set. Given this is a multi-class prediction problem, we train using a sigmoid function at the output of our network to predict a set of candidate parameter configurations. At test time we evaluate with the parameters predicted by the maximally activated output unit. Our results are presented in tables 3.2, and 3.3.

We used the pretrained Inception V3 [46] and InceptionResNet V2 [46] ImageNet classifiers as our source networks. To generate adversarial examples on these networks, we used the open-source Cleverhans toolbox [20]. the model was trained using SGD with Nesterov momentum for 25 epochs. We then test on six different validation sets, one for each adversary respectively. We show (A) the top-5 accuracy of recovering the original predicted classification label from the adversarial example (note this is not necessarily the ground truth label), as well as (B) how often AF is able to defeat the adversarial attack - changing the prediction from the adversarial label to a new one.

Source Network	Clean	FGSM	PGD	MIM	CW	DeepFool	L-BFGS
AF+Inc $V3_A$	95.0	89.0	90.7	79.1	89.1	90.3	81.3
AF+Inc $V3_B$	95.0	95.9	98.0	96.4	94.1	95.3	96.2
AF+IncResNet $V2_A$	91.1	87.2	87.1	75.3	87.8	85.0	80.8
AF+IncResNet $V2_B$	91.1	93.1	98.0	94.3	97.8	92.0	95.3

Table 3.2: Percentage of adversarial examples detected by the adaptive bilateral filtering (AF) network across different attacks.

It can be seen that we recover adversarial examples generated by FGSM, PGD, CW and DeepFool near perfectly, while missing nearly 15% of the examples of MIM and L-BFGS. These results are significantly better than the results in [28], which used a 3x3 average filter to recover images. Our Adaptive Filtering network succeeds in removing adversarial examples generated on natural images, with a relatively simple network. This makes the Adaptive Filtering network a viable method for defending networks against an adversary who employs a wide range of attacks.

Network	Clean	FGSM	PGD	MIM	CW	DeepFool	L-BFGS
Inc V3 $_{t1}$	78.8	30.1	0.2	0.1	0.1	0.7	0.0
Inc V3 $_{t5}$	94.4	65.2	4.8	5.5	7.3	0.5	12.1
AF+Inc V3 $_{t1}$	71.7	71.0	71.6	63.1	71.1	70.1	64.2
AF+Inc V3 $_{t5}$	89.6	84.0	86.3	74.6	84.1	85.2	76.7
IncResNet V2 $_{t1}$	80.4	55.3	0.8	0.5	0.3	2.5	0.0
IncResNet V2 $_{t5}$	95.3	72.1	15.8	10.2	10.3	8.5	19.2
AF+IncResNet V2 $_{t1}$	73.1	70.1	70.8	60.5	70.3	70.5	65.0
AF+IncResNet V2 $_{t5}$	86.7	83.1	82.8	71.7	83.6	85.6	77.0

Table 3.3: Top-1 and top-5 accuracy of InceptionV3 and Inception-ResNetV2 on adversarial examples.

3.4.2 BFNet Defending Against Counter Attacks on ImageNet

Due to the high cost of adversarial training on natural images, on ImageNet we perform only one round of counter-attack. We assume that the attack knows about BFNet and attack it by backpropagating through the entire BFNet defense. We use this as an opportunity to test the robustness of BFNet that cannot be attributed to adversarial training. To this end, we use the Inception V3 and the Inception-ResNet V2 networks, and add our bilateral filter layer to the input, keeping the pretrained ImageNet weights. We test against both L_2 and L_∞ adversaries to obtain a complete picture of the robustness of BFNet. L_∞ is a more informative metric when discussing the magnitude of adversarial attacks on very small images, because a large perturbation measured under the L_∞ norm equates to a large visual change across few pixels. But with large natural images, a perturbation with a large L_∞ distance is less interpretable. A large change to a single pixel may still go unnoticed to a human observer, while a large perturbation under the L_2 norm gives more information about the *total distortion* caused by the adversarial attack.

To measure resistance to attacks under the L_2 norm, we use the unbounded attacks L-BFGS and DeepFool. It is impossible to be fully resistant to unbounded attacks, because any image can be changed to a completely different image and its CNN output would certainly change. Hence, we report the average L_2 and L_∞ distance of the adversarial images to the original ones from the unbounded attacks. From Table 3.4 we can see that

(a) Adversarial examples generated by L-BFGS on a BFNet version of the Inception V3 classifier. Generated adversarial examples have visually identifiable perturbations, and have an average L_2 norm of 106.2

(b) Adversarial examples generated by DeepFool on a BFNet version of the Inception V3 classifier. The generated adversarial examples have large, noisy perturbations, and have an average L_2 norm of 181.2

Figure 3.2: Adversarial images created with BFNet.

our approach yields a very robust model against adversarial perturbations under the L_2 metric. When attacking our BFNet models with DeepFool, we see that the generated adversarial image has an L_∞ distance over *30x larger*, when compared to an unmodified network of the same architecture. Similarly, we can see that the L_2 distance of an adversarial generated against BFNet is far larger when compared to adversarial images generated against a network of the same architecture without the bilateral filter. With respect to the L-BFGS attack, we see a similarly large disparity between BFNet and a vanilla network. Fig.3.2 shows some examples of images generated by those adversarial counterattacks. One can see that the DeepFool and LBFGS attacks had to significantly modify the image to defeat BFNet, creating clearly visible patterns. An adversarial detector or a human eye would easily be able to detect those attacks.

For the L_∞ attacks such as FGSM, and MI-FGSM we measure the resistance of our model to different values of perturbation ϵ. We can see that our BFNet significantly

Network	DeepFool		L-BFGS	
	L_∞	L_2	L_∞	L_2
Inception $V3_{\text{Natural}}$	0.015	0.43	0.02	0.67
Inception $V3_{\text{BFNet}}$	0.621	148.29	0.39	90.52
IncResNet $V2_{\text{Natural}}$	0.025	0.44	0.06	0.77
IncResNet $V2_{\text{BFNet}}$	0.793	187.45	0.65	90.65

Table 3.4: Performance of BFNet against DeepFool and L-BFGS attacks.

decreases the attack strength of L_∞ adversaries, in most cases by over 50%. Of particular note is that we show more significant resistance to adversarial perturbations of $\epsilon \leq 0.3$. For both attacks we use 1,000 random images sampled from the ILSVRC 2012 validation set, and report the percentage of successful attacks against the natural model and BFNet respectively.

Network	Epsilon	FGSM		MI_FGSM	
		Natural	BFNet	Natural	BFNet
Inception V3	0.1	73.2	**30.2**	58.8	**21.0**
Inception V3	0.15	78.6	**36.6**	65.6	**30.1**
Inception V3	0.3	93.2	**46.6**	88.2	**42.2**
Inception V3	0.5	99.0	**63.2**	98.0	**52.4**
Inception V3	0.75	100.0	**90.8**	99.6	**53.4**
Inception V3	1.0	100.0	**99.8**	99.6	**70.8**
IncResNet V2	0.1	58.8	**42.6**	89.1	**59.0**
IncResNet V2	0.15	65.6	**55.4**	90.6	**38.6**
IncResNet V2	0.3	88.2	**75.4**	92.2	**59.4**
IncResNet V2	0.5	98.0	**91.0**	100.0	**74.6**
IncResNet V2	0.75	**99.6**	**99.6**	100.0	**80.2**
IncResNet V2	1.0	**99.6**	**99.6**	100.0	**91.8**

Table 3.5: Performance of BFNet against FGSM and MI-FGSM adversaries for a range of perturbation sizes (lower is better).

3.4.3 Adversarial Training

Finally, we experiment with adversarial training with BFNet on the MNIST and CIFAR-10 datasets.

3.4.3.1 MNIST

To show that BFNet is robust to strong first-order adversaries, we train a small CNN to 99.2% accuracy on the test set. Our model consists of 2 convolutional layers with 32 and 64 filters respectively, each followed by 2 x 2 max pooling and ReLU. We use a final fully connected layer with 1024 units. We modify our network into a BFNet by adding our bilateral filter layer at the input of the first convolutional layer. We then train with adversarial training using three distinct adversaries: FGSM, PGD, and PGD with the proposed CW loss function. We report the results in table 3.7. Our results perform well against the state-of-the-art adversarial training results. We also show that when our network is trained on a single strong adversary, we are robust to attacks from other adversaries.

Network	Clean	FGSM	PGD	CW	CW (κ=50)
$\text{BFNet}_{\text{pgd}}$	**99.0**	95.5	**98.0**	93.2	-
$\text{BFNet}_{\text{fgsm}}$	**99.0**	**98.1**	36.4	88.2	**96.0**
Madry	98.8	95.6	93.2	**94.0**	93.9
Tramer_{A}	98.8	95.4	96.4	-	95.7
Tramer_{B}	98.8	97.8	-	-	-

Table 3.6: Comparison of our method with state of the art adversarial training results on MNIST.

During training we observe a faster convergence in training loss (see Appendix), and increased robustness to white-box FGSM, PGD, and CW attacks, when trained against only the PGD attack. However, the model trained against FGSM does worse against stronger adversaries such as PGD, as the attack itself is a weak adversary.

Network	Type	Clean	FGSM	PGD
$\text{BFNet}_{\text{pgd}}$	A	87.1	55.2	**50.4**
$\text{BFNet}_{\text{pgd}}$	B	73.1	64.5	38.1
$\text{BFNet}_{\text{fgsm}}$	B	76.5	**70.6**	12.2
Madry	A	87.3	56.1	45.6

Table 3.7: Performance of our two adversarially trained BFNets on CIFAR-10.

3.4.3.2 CIFAR10

We perform similar experiments to test BFNet on CIFAR-10. We use a network with four convolutional layers, each followed by 2x2 max pooling. A linear layer of $4,096$ units is used before softmax. For BFNet, a differentiable bilateral filter layer to preprocess the images. When naturally trained with Adam for 30 epochs, this network reaches an accuracy of 79.04% on the test set. We also train the original ResNet-18 model used in [32] for 80K iterations. Trained on natural examples we reached an accuracy of 92.7% on the test set. Each model is then trained adversarially with PGD and FGSM, respectively. We use an L_∞ bound of $\epsilon = 8$ for both adversaries. We use 20-step PGD with a learning rate of 2.0. We report our results in Table 3.7 and it can be seen that our BFNet trained on PGD outperforms [32] significantly on the PGD adversary.

In contrast to MNIST, CIFAR-10 remains a very challenging dataset. The higher dimensionality makes robust training significantly more difficult. Because CIFAR-10 is too small, the edges are not so obvious, which could have hurt our performance. We believe that the bilateral filtering poses more constraint in larger natural images, such as ImageNet.

3.5 Discussion

The continued existence of adversarial examples, and the lack of effective defenses limits our ability to deploy AI systems in critical areas where safety and security are necessary. In this paper we showed that a bilateral filter can be used as a core part of versatile, effective defenses to recover clean images from perturbed ones. The bilateral filter remains effective when deployed with numerous defense strategies: as a manual preprocessing step, a trained denoiser, or a robust model that is trained end-to-end. Our

defense holds in multiple attack settings where the attacker has knowledge about it. In the future we hope to explore even better filter approaches which projects even better to the natural image manifold and limit adversarial examples, as well as combining it with an adversarial detection approach to construct a comprehensive defense.

Because the bilateral filter encourages piece-wise smoothness, we see that the bilateral filter effectively projects adversarial images back to the distribution of natural images. Furthermore, when trained end to end, our bilateral filter can be combined with adversarial training approaches to create a robust defense method. In the future we hope to see preprocessor defenses tested by direct white-box attack, and combined with robust optimization methods like adversarial training to create performant, unified defenses.

Chapter 4: Conclusion

In this thesis we examined two problems with using neural networks as the main estimation tool in a machine learning system. We saw that neural networks do not provide well-calibrated uncertainty estimates on their predictions by default. Indeed the maximum likelihood principle gives us no guarantees that the predictions and their associated probabilities resemble anything meaningful. We also saw that extreme forms of out-of-distribution data called adversarial examples pose a threat to critical systems where safety is a concern. By solving a simple optimization problem, an attacker can force a neural network to behave in ways counter to their training.

To handle out-of-distribution data we proposed HyperGAN. We showed that HyperGAN can quickly generate diverse ensembles of neural networks. These ensembles can be as large as desired and can be generated quickly, with one forward pass per member. Larger ensembles more closely approximate the distribution over parameters learned by HyperGAN, hence they offer a more comprehensive prediction of uncertainty with respect to the larger data density learned by HyperGAN. With large ensembles generated by HyperGAN, we showed that we can reliably detect out-of-distribution data including adversarial examples.

We also explored robustness of neural networks in the context of adversarial examples. We proposed using a bilateral filter as a flexible defense against adversarial examples. When used as a preprocessor, a bilateral filter with the right parameters can recover the original input beneath the perturbed image. Bilateral filtering also works in a dynamic threat model. Adapting a bilateral filter as a trainable layer in a neural network acts as a projection on each input, forcing it onto the natural image manifold. When combined with adversarial training, the resulting model performed well against white-box direct attacks.

These two methods comprise but small steps in the large effort to create neural networks which are fundamentally reliable. In this respect there is far more work to do. The largest problem currently is the scalability of *any* robust learning algorithm. All methods which are robust to adversarial examples, or provide uncertainty estimates

are not tenable in very high dimensional domains. HyperGAN requires at least a $100x$ increase in parameters to operate, while normalizing flows often sport an even larger multiplier. Robustness to strong adversarial attack comes at the high cost of adversarial training, which vastly increases training time. Running adversarial training properly requires 40+ iterations of PGD per training example seen. These shortcomings are important to consider if we consider the wide variety of applications that neural networks are being deployed in.

Bibliography

[1] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast High-Dimensional Filtering Using the Permutohedral Lattice. *Computer Graphics Forum*, 2010.

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016.

[4] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A Closer Look at Memorization in Deep Networks. *ArXiv e-prints*, June 2017.

[5] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[6] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016.

[7] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[9] Andreas Damianou and Neil Lawrence. Deep Gaussian processes. In C. Carvalho and P. Ravikumar, editors, *Proceedings of the Sixteenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, AISTATS '13, pages 207–215. JMLR W&CP 31, 2013.

[10] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with JPEG compression. *CoRR*, abs/1705.02900, 2017.

[11] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[12] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Discovering adversarial examples with momentum. *CoRR*, abs/1710.06081, 2017.

[13] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M. Roy. A study of the effect of JPG compression on adversarial images. *CoRR*, abs/1608.00853, 2016.

[14] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *CoRR*, abs/1707.08945, 2017.

[15] C. D. Freeman and J. Bruna. Topology and Geometry of Half-Rectified Network Optimization. *ArXiv e-prints*, November 2016.

[16] Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *ArXiv e-prints*, June 2015.

[17] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *ArXiv e-prints*, December 2014.

[19] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[20] Ian J. Goodfellow, Nicolas Papernot, and Patrick D. McDaniel. cleverhans v0.1: an adversarial machine learning library. *CoRR*, abs/1610.00768, 2016.

[21] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016.

[22] J. M. Hernández-Lobato and R. P. Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. *ArXiv e-prints*, February 2015.

[23] Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

[24] Markus Holopainen and Peter Sarlin. Toward robust early-warning models: A horse race, ensembles and model uncertainty. *Quantitative Finance*, 17(12):1933–1963, 2017.

[25] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[26] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian Hypernetworks. *ArXiv e-prints*, October 2017.

[27] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *ArXiv e-prints*, December 2016.

[28] Xin Li and Fuxin Li. Adversarial examples detection in deep networks with convolutional filter statistics. *CoRR*, abs/1612.07767, 2016.

[29] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Jun Zhu, and Xiaolin Hu. Defense against adversarial attacks using high-level representation guided denoiser. *CoRR*, abs/1712.02976, 2017.

[30] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *CoRR*, abs/1605.09673, 2016.

[31] Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257, 2011.

[32] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ArXiv e-prints*, June 2017.

[33] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.

[34] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. *CoRR*, abs/1705.09064, 2017.

[35] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[36] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.

[37] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *CoRR*, abs/1703.00837, 2017.

[38] Nicolas Papernot and Patrick D. McDaniel. On the effectiveness of defensive distillation. *CoRR*, abs/1607.05113, 2016.

[39] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 372–387, 2016.

[40] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508, 2015.

[41] Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International Conference on Machine Learning*, pages 324–333, 2016.

[42] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[43] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *CoRR*, abs/1805.06605, 2018.

[44] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *CoRR*, abs/1511.05432, 2015.

[45] Shiwei Shen, Guoqing Jin, Ke Gao, and Yongdong Zhang. AE-GAN: adversarial eliminating with GAN. *CoRR*, abs/1707.05474, 2017.

[46] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR 2016 Workshop*, 2016.

[47] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[48] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein Auto-Encoders. *ArXiv e-prints*, November 2017.

[49] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[50] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Deep image prior. *CoRR*, abs/1711.10925, 2017.

[51] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

APPENDICES

Appendix A: Appendix: HyperGAN

We show the first filter in 25 differernt networks generated by the HyperGAN to illustrate their difference in Fig. A.2. It can be seen that qualitatively HyperGAN learns to generate classifiers with a variety of filters.

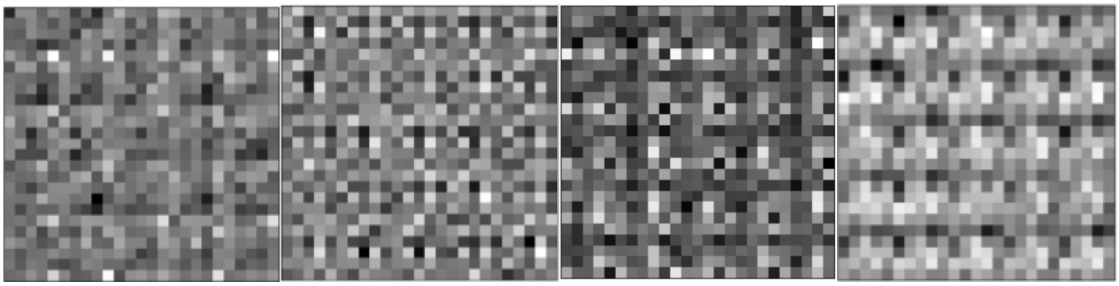


Figure A.1: Convolutional filters from MNIST classifiers sampled from HyperGAN. For each image we sample the same 5x5 filter from 25 separate generated networks. From left to right: figures a and b show the first samples of the first two generated filters for layer 1 respectively. Figures c and d show samples of filters 1 and 2 for layer 2. We can see that qualitatively, HyperGAN learns to generate classifiers with a variety of filters.

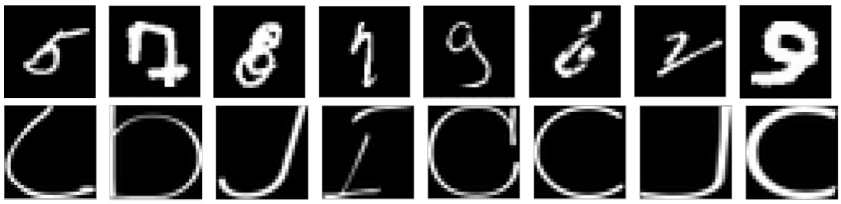


Figure A.2: Images of examples which do not behave like most of their respective distribution. On top are MNIST images which HyperGAN networks predict to have high entropy. We can see that they are generally ambiguous and do not fit with the rest of the training data. The bottom row shows notMNIST examples which score with low entropy according to HyperGAN. It can be seen that these examples look like they could come from the MNIST training distribution, making HyperGAN's predictions reasonable

Appendix B: Appendix: Robust Neural Networks

B.0.1 More examples of bilateral filtering results on adversarial images

Fig. B.1 shows more examples of adversarial images from several different attacks after bilateral filtering.

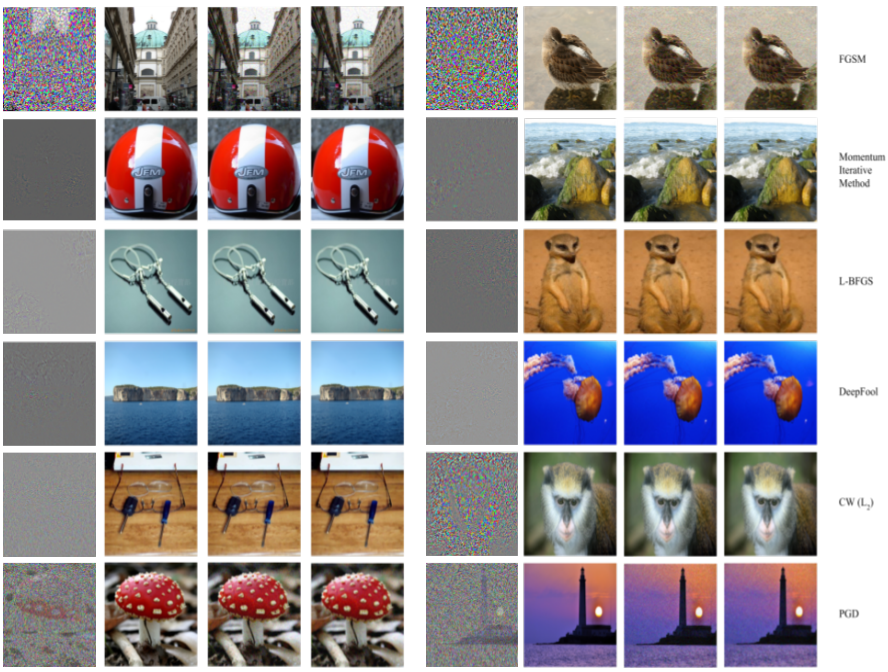


Figure B.1: The effect of bilateral filtering on adversarial inputs. From left to right, we show the adversarial perturbation, the clean image, the adversarial images generated by the respective attack algorithms, and the recovered image after bilateral filtering. Note that bilateral filtering does not destroy image quality, and images can be correctly classified.

B.0.2 Adversarial images without BFNet

Fig. B.2 shows the images generated with DeepFool and L-BFGS on ImageNet without BFNet added to preprocess the images. Compared with Fig. 3.2, these adversarial images are indiscernible with real ones from human eyes.

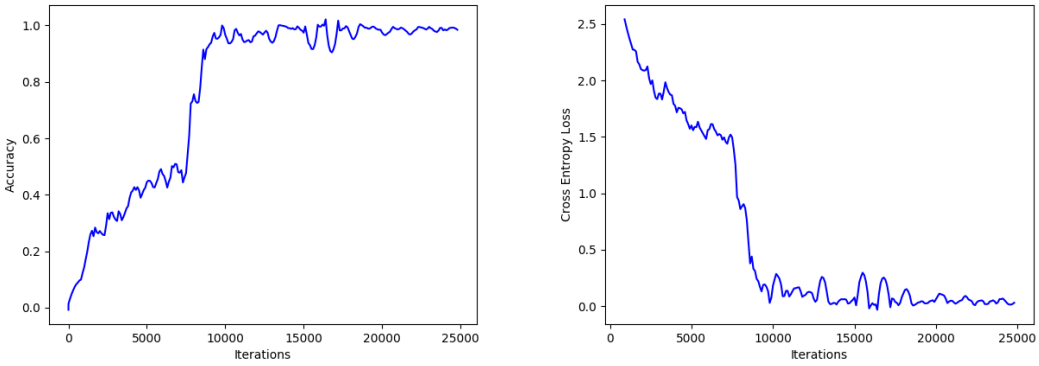
(a) Adversarial images generated with DeepFool on a vanilla Inception V3 classifier. The adversarial images are visually identical to the real images, and have an average L_2 norm of 0.09

(b) Adversarial examples generated by an L-BFGS adversary on a vanilla Inception V3 classifier. The adversarial examples have an average L_2 distance of 0.025 from their natural counterparts, and have visually imperceptible perturbations.

Figure B.2: Adversarial images generated by DeepFool and L-BFGS without BFNet, to be compared with Fig.3.2

B.0.3 Convergence of the adversarial training

Fig. B.3 shows the convergence of the adversarial training component of training BFNet.



(a) (b)

Figure B.3: The average mini-batch batch accuracy (a) and cross entropy loss (b) for the model trained with adversarial training on MNIST. We trained our model to convergence, which happened near 20k iterations. We can see that the training is stable and converges to a similar training error as a naturally trained network

B.0.4 Failure examples for adversarially trained BFNet

Fig.B.4 and Fig.B.5 showed all the failure images after adversarial training on the MNIST dataset.

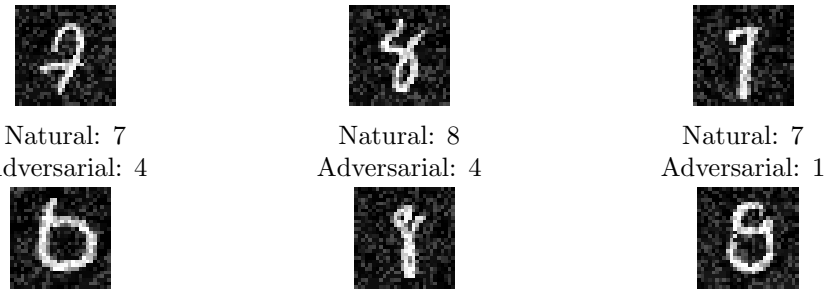


Figure B.4: PGD adversarial examples which fool an adversarially trained $\mathrm{BF}_{\mathrm{PGD}}$ with $\epsilon = 0.3$

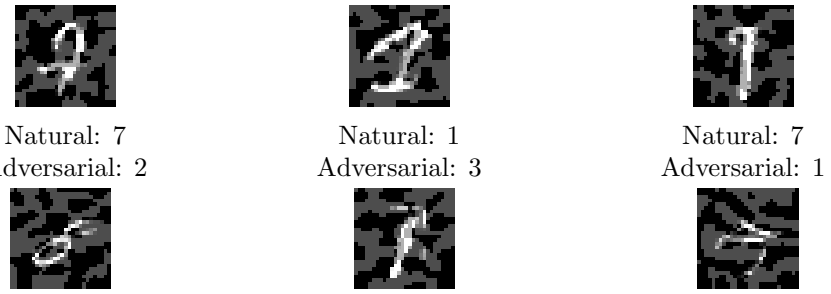


Figure B.5: FGSM adversarial examples which fool adversarially trained $\mathrm{BF}_{\mathrm{fgsm}}$ with $\epsilon = 0.3$