A Survey of Bayesian Neural Networks against Adversarial Attacks

Michael Przystupa, Marjan Albooyeh, Clement Fung

April 23, 2018

1 Abstract

We explored the ability of different Bayesian neural network in detecting uncertainty, when provided adversarial images in both colour and greyscale. Our findings suggest that although a multitude of variational approaches exist to approximate the posterior distribution of a Bayesian neural network, some methods are more effective at measuring uncertainty when faced with unfamiliar inputs compared to others. Bayes by Backprop and MC dropout were the most robust in measuring uncertainty when faced with greyscale adversarial images, while probabilistic back propagation showed to be the least effective and effectively behaved similarly to a non-Bayesian neural network. Our findings on color images were less pronounced. These findings suggest that these approaches to approximating a posterior distribution in the Bayesian approach to neural networks are likely more effective for real-world image detection problems.[1]

2 Introduction

As neural networks become deeper and more complex, image classification has started to reach the point of being a solved task [4]. With this realization, societal reliance on image classification tasks has also grown correspondingly: Face detection has become a form of authentication, text-reading is automated, and cars are driving themselves.

Despite our growing ability to perform these tasks, apparent risks are still present with their usage. Recent work has shown the brittleness of modern image classifiers: by perturbing the value of a pixels in a test image, classification output probabilities are significantly modified, such that class predictions are changed. These test images are known as **adversarial examples**, and they pose a major risk to the robustness of image classifiers. Adversarial examples are created through the combination of a normal image with a perturbation, such that the human eye is unable to notice the difference between an adversarial example and a genuine one [8]. These attacks have extended to reach beyond the digital modification of test examples, and even an attack using dark tape on road signs has been shown to fool self-driving vehicles [5]. Fundamentally, these examples are difficult to detect, both by humans and machine learning models. Deep neural network image classifiers are notoriously complex and their decision boundaries are highly irregular. Knowing this, adversaries are able to repeatedly

[1] For our code and more plots, please see the github repository at: https://github.com/gamerDecathlete/AdversarialDetectionWithBNNs

evaluate the decision boundaries of these models and discover regions that fool them.

One solution to this problem, DeepXplore [16] provides a testing suite that compares the decision boundaries across multiple image classifiers and reports these regions. DeepXplore then updates the parameters of these models such that these regions are no longer anomalous. Fundamentally, this solution is only a proof by example, as it relies on the existence of multiple image classifiers and can only report regions on which they disagree.

A variety of other ML detection techniques have also been attempted for detecting adversarial examples, but have been rejected as ineffective [3]. Some of these defenses rely on neural networks to detect adversarial examples, but it seems that any neural network can be fooled with some adversarial example, rendering these techniques ineffective.

Bayesian models, a class of probabilistic graphical models, provide a way to reason about uncertainties and express a model's confidence in its predictions. In other words, the model can express how confident it is in making predictions, which can be useful for critical applications in which an important decision has to be made and should provide additional robustness against adversarial examples.

A Bayesian Neural network (BNN) is a neural net with a probability distribution over the network weights, which enable the network to express uncertainty regarding predictions. In these models, we update the prior belief on the weights based on observed data. A prediction is then typically made with the following equation:

$$P(y^*|x^*) = \int p(y^* \mid x^*, W, D) * P(W \mid D)dw \tag{1}$$

Where y^* and x^* are new data, D is previously seen data and W is our network parameters. This is computationally intractable, but an approximation can be made by sampling from the posterior distribution repeatedly and summing the results to obtain a distribution for each prediction.

These models allow us to measure the confidence in the predictions of our neural network, providing a potential way to detect and deal with adversarial attack if the variance of the outputs for an image is particularly high. The literature exploring BNNs in this capacity is quite limited having only otherwise been explored in [18].

In this work, we attempt to recreate the results in [18], and extend their work to additional adversarial attacks. We also build on their suggested future work by applying the same adversarial attacks to colored images to demonstrate the effectiveness on images with multiple input channels. We add an additional BNN approach previously not explored in their work, which uses a blackbox variational inference optimization approach to maximize the evidence lower Bound (ELBO) loss to compare these other more novel BNN approaches.

3 Background

3.1 Adversarial Examples

There are two classes of adversarial examples: white box examples, in which an adversary is able to observe all the information of a model such the weights and loss functions, and blackbox examples, in which an adversary only has access to prediction APIs. Whitebox attacks are, by definition, a more powerful class of attack, and thus we aim to defend against these types of examples. A variety of ML detection techniques, such as PCA, neural nets, and kernel density estimation, have also been attempted for detecting both whitebox and blockbox adversarial examples, but have been rejected as ineffective [3]. We propose the exploration of three adversarial example generation techniques, mostly because they represent the state of the art, and are also implemented in the cleverhans software library [14].

FGSM Method [8] first proposed the Fast Gradient Sign Method (FGSM) as a technique for generating adversarial examples. The method proposes using gradient steps to find adversarial examples, where steps η are taken in the direction that pushes examples toward decision boundaries. For a defined loss function $J(\theta, x, y)$, where x and y represent the training data and θ represents the fixed model parameters:

$$\hat{x} = x + \epsilon(sign(\nabla_x J(\theta, x, y)))$$

simply taking a step of size $\epsilon = 0.25$ in the direction of the gradient causes ImageNet to produce errors of 90%.

JSMA Method An alternative method for producing adversarial examples, known as the Jacobian-based Saliency Map Attack (JMSA) [15], involves performing a greedy attack, in which the algorithm scans over an adversarial example, modifying pixels one at a time to achieve a targeted class.
At each pixel, the attack involves generating a saliency map, which models the effect each pixel's value has on the resulting classification when a step is taken in direction of the forward derivative of the network. For pixels with a high saliency value, modifications will increase the likelihood of a misclassification. JSMA iteratively modifies pixels until a resulting misclassification is produced, up to a predefined total budget γ, which bounds the total maximum L2 norm of the perturbation step.

Gaussian Noise In [21] the authors demonstrated that a single pixel perturbation could be enough to throw an image classifier off with 70% success. Given the simplicity of such attacks, it suggests that an adversary can perform relatively simple transformations to an image. Although other works have shown that incorporating such examples into the data set can improve the robustness of a neural network, such methods do not solve the problem of of classifying

examples under new perturbations which the network was not trained on. With this observation, we follow [18] and apply a Gaussian noise mask with increasing variance as an adversarial attack to measure how well BNNs deal with simple image transformations, Gaussian noise also serves as a baseline example of the simplest attack known, which has no knowledge about the model internals.

3.2 Variational Inference for Bayesian Neural Networks

One challenge with BNNs is the intractability of calculating the posterior distribution over the weights. A popular approach to approximate this is variational inference, for which a number of authors have produced algorithms to learn an approximate variational posterior distribution [13, 9, 10, 12, 7, 2].

Practical Variational Inference for Bayesian Neural Networks Variational methods for creating Bayesian neural nets have only been applicable to a few simple network architectures. This is largely due to the difficulty of deriving analytic solutions to the required integrals over the variational posteriors. [9] proposed an approach which searches for variational distributions whose expectation values can be efficiently approximated with numerical integrals instead of finding analytic solutions. The result is a stochastic method for variational inference that can be used to build BNNs, which reformulates inference as an optimization problem that can easily be implemented in existing neural network software.

For complex networks, it's hard to calculate the exact posterior analytically, therefore the variational inference approach approximates the actual posterior $P(w|D,\alpha)$ with a more tractable distribution $Q(w|\beta)$ and tries to minimize the difference between these two. In this approach, the loss function contains the loss and KL divergence between the actual and approximated distributions so one can control model accuracy and model complexity at the same time. This loss function can be reinterpreted as a *minimum description loss function* and has the form

$$L(\alpha,\beta,D) = \mathbb{E}_{w\sim Q(\beta)}[L^N(w,D)] + D_{KL}(Q(\beta)||P(\alpha)) \tag{2}$$

where $L^N(w,D)$ is the negative log likelihood and $D_{KL}(Q(\beta)||P(\alpha))$ is the Kullback-Leibler divergence between $Q(\beta)$ and $P(\alpha)$.

As a choice of distribution for prior and posterior, the authors used the Delta, Laplace and Gaussian distributions. Based on the results, the networks with Gaussian priors and posteriors outperformed others. In this case, the derivatives of the loss function with respect to μ_i and σ_i^2 will be

$$\frac{\partial L(\alpha,\beta,D)}{\partial \mu_i} \approx \frac{\mu_i - \mu}{\sigma^2} + \sum_{(x,y)\in D} \frac{1}{S}\sum_{k=1}^{S} \frac{\partial L^N(w^k,x,y)}{\partial w_i} \tag{3}$$

$$\frac{\partial L(\alpha,\beta,D)}{\partial \sigma_i^2} \approx \frac{1}{2}\Big[\frac{1}{\sigma^2} - \frac{1}{\sigma_i^2}\Big] + \sum_{(x,y)\in D} \frac{1}{2S}\sum_{k=1}^{S} \Big[\frac{\partial L^N(w^k,x,y)}{\partial w_i}\Big] \tag{4}$$

where a separate set of S weight samples $\{w^k\}_{k=1}^S$ is drawn from $Q(\beta)$ for each (x, y) and S is the number of Monte-Carlo samples.

Using a Gaussian prior with fixed variance is equivalent to adding zero-mean, fixed-variance Gaussian noise to the network weights during training. The model will be evaluated on test data by removing weight noise with *maximum a posteriori* approximation $P(y|x, w^*)$, where w^* is the mode of $Q(\beta)$.

Variational Matrix Gaussian In [12] the authors propose treating each layer of a BNN as a matrix variate Gaussian distribution. This parameterization allows one to separate the correlations among rows and columns of the matrix. The posterior can be optimized via the KL-Divergence by approximating the covariance matrix as a diagonal matrix, and a local parameterization trick that has been shown to be a formulation of a deep Gaussian process for more effective sampling. The authors showed in their experiments that VMG posteriors worked better than [10] in 8 of 10 regression tasks and better on the classification for the MNIST dataset.

Bayes By Backprop [2] introduces an efficient variational approach for regularization built upon Bayesian inference on the weights of the network, which prevents overfitting and improves predictive performance. This approach assumes the variational posterior as a diagonal Gaussian distribution and prior as a mixture of two Gaussian densities. Both Gaussian distributions have zero mean and different variances. When classifying MNIST digits, performance from Bayes by Backprop is comparable to that of dropout.

Probabilistic Back Propagation Probabilistic Back propagation is a Bayesian training technique which is similar to back propagation in typical neural networks and was proposed in [10]. In the forward pass, the data is fed through the network and each weight in the network is approximated as Gaussian distribution which are then used to generate the output, calculating the logarithm of the probability of the target variable. These values are then used to generate the gradients passed backward through the network to update the Gaussian marginal distributions. This requires twice as many parameters than regular neural networks by having incorporating a Gaussian on each weight dimension, but allows for direct sampling over the distribution of the output.

MC Dropout Method All the above approaches can be used to reason about model uncertainty, but they come with a prohibitive computational cost. [7] shows that a deep neural network, with arbitrary depth, non-linearity, and probabilistic dropout applied before every weight layer, is mathematically equivalent to an approximation of a deep Gaussian process model and can represent model uncertainty in deep learning without adding extra computation cost. With dropout, we sample binary variables which takes value 1 with probability p_i for each layer i (except the last layer). A unit in a layer is dropped for a given input if its corresponding binary variable takes value 0. The same values will

be used in the backward pass propagating the derivatives to the parameters. In this model the expectation of approximate predictive distribution $q(y^*|x^*)$ is

$$\mathbb{E}_q(y^*|x^*)((y^*)^T(y^*)) \approx \tau^{-1} I_D + \frac{1}{T}\sum_{t=1}^{T} \hat{y}^*(x^*, W_1^t, ..., W_L^t)^T \hat{y}^*(x^*, W_1^t, ..., W_L^t) \tag{5}$$

Where y^* and x^* are test examples and we sample T sets of vectors from the Bernoulli distribution $\{z_1^t, ..., z_L^t\}_{t=1}^T$ giving $\{W_1^t, ..., W_L^t\}$. τ is the model precision and $\hat{y}^*$ is random variable realization. Also, a model's predictive variance will be as follows

$$\begin{aligned} Var_{q(y^*|x^*)}(y^*) \approx \tau^{-1} I_D & \\ + \frac{1}{T}\sum_{t=1}^{T} \hat{y}^*(x^*, W_1^t, ..., W_L^t)^T \hat{y}^*(x^*, W_1^t, ..., W_L^t) & \\ - \mathbb{E}_{q(y^*|x^*)}(y^*)^T \mathbb{E}_{q(y^*|x^*)}(y^*) & \end{aligned} \tag{6}$$

Obtaining model uncertainty with dropout is equivalent to sampling variance of T stochastic forward passes through the neural network (which is known as *Monte Carlo dropout*) plus the inverse model precision. In other words, to estimate the predictive mean and predictive uncertainty, we average the results of stochastic forward passes.

4 Previous Work

4.1 Adversarial attacks on Bayesian Neural Networks

[18] provides an extensive study which used four different techniques to train a BNN and measure the uncertainty of predictions caused by two type of adversarial approaches. This included FGSM [8] and Gaussian perturbations in the image. They found that a BNNs accuracy was negatively affected by these two particular types of attacks like regular neural networks, but showed increased measurements of uncertainty in their results, suggesting that BNNs are indeed more robust to adversarial examples.

To the best of our knowledge, this is the only study exploring the application of BNNs in this setting. The authors leave a number of potential extensions to their work, including exploring the effects of adversaries on colored images and exploring MC dropout more extensively. They do not raise addressing other adversarial types seen in other works such as [5, 23]. Our work extends their work by attempting to replicate their results on MNIST, adding an additional model and adversary type, as well as applying these methods to CIFAR10, which is a colored images dataset.

5 Approach

In this section we highlight the details to each BNN approach we applied and offer some configuration details we used. We then explain the adversaries we generated for each dataset and how they are passed into our BNNs, and discuss the metrics we used to measure uncertainty when training our models.

5.1 Bayesian Neural Nets

5.1.1 MC Dropout

We approximate our model's intractable posterior with Bernoulli variational distributions, which is similar to dropout layers in CNNs. We train a LeNet-like CNN with a dropout layer after convolution layers for 200 epochs. We test dropout rates in range $[0.2, 0.7]$ and we get our best results with dropout rate 0.5. In test time, for each adversarial example, we run $T = 100$ forward passes through the network and compute uncertainties and predicted probabilities.

5.1.2 Probabilistic Backpropagation

The Probabilistic backpropagation algorithm minimizes the KL divergence between the previous beliefs $q_{old}(w)$ of the weights $s(w) = Z^{-1} f(w) N(W \mid m, v)$, where Z is a normalization constant, $f(w)$ is a likelihood function and $N(W \mid m, v)$ is the normal distribution. Using these quantities, the authors show that the update can be written as:

$$m^{new} = m + v \frac{dlog(Z)}{dm} \tag{7}$$

$$v^{new} = v - v^2 [\frac{dlog(Z)}{dm}^2 - 2 * \frac{dlog(Z)}{dv}] \tag{8}$$

We use the implementation of probabilistic backpropgation from [10]. We used 3 hidden layers, using the author's recommended hidden layer size of 40 units, trained for 50 epochs. This implementation is designed for regression tasks, but still worked well on CIFAR10 and MNIST. To obtain our labels we rounded the outputs to the nearest integer value and then clamped the value in the numerical representation of our classes (0 - 9 for both data sets). To obtain a probability distribution over the outputs, we sample 10 outputs of the network which we use to count the number of times different labels are predicted for each image. We then run this through a softmax function to get the posterior distribution like in the other approaches, and repeated this proceedure 50 times to perform our analysis.

5.1.3 Bayes by Backprop

[2] utilize a proposition where the derivative of an expectation can be expressed

as the expectation of a derivative:

$$\frac{d}{d\theta}E_{q(w|\theta)}[f(w,\theta)] = E_{q\epsilon}[\frac{df(w,\theta)}{dw}\frac{dw}{d\theta} + \frac{f(w,\theta)}{d\theta}] \tag{9}$$

Using this formulation, it allows one to calculate the variational parameters by calculating the updates for the mean μ and variance ρ as :

$$\mu = \mu - \alpha\frac{df(w,\mu)}{dw}\frac{dw}{d\mu} + \frac{f(w,\mu)}{d\mu} \tag{10}$$

$$\rho = \rho - \alpha\frac{df(w,\rho)}{dw}\frac{dw}{d\rho} + \frac{f(w,\rho)}{d\rho} \tag{11}$$

We modified an existing implementation: which contained two hidden layers with 800 units each, using the mixture of Gaussian prior proposed [2]. We trained the model for 100 epochs on batch sizes of 512 value using the Adam [11] optimizer with an initial learning rate of 10^{-4}. For testing, we sampled the posterior distribution 100 times to generate our performance metrics.

5.1.4 Pyro

Pyro is a probabilistic programming language designed based on a collection of results to perform variational inference over a directed acylic graphical model [1, 19, 17, 22], and is most akin to the approach of [9]. We used a single hidden layer network with 1200 hidden units and applied a Gaussian distribution initialized with random parameters near 0 for our variational distribution $q(w \mid \mu, \sigma)$ to each layer of our neural network. Our prior was a relatively wide Gaussian distribution $p(w) = N(w|0, 10)$. We then optimized this network using the evidence lower bound [22] with the Adam optimizer [11] with an initial learning rate of 10^{-4} training for 150 epochs using batches of 1024, and collected 100 samples from the posterior distribution to perform our analysis.

5.1.5 VMG

[12] defines a matrix variate Gaussian as a three parameter distribution that governs a random matrix W:

$$\begin{aligned} p(W) &= MN(M,U,V) \\ &= \frac{\exp(-\frac{1}{2}tr[V^{-1}(W-M)^TU^{-1}(W-M)])}{(2\pi)^{np/2}|V|^{n/2}|U|^{n/2}} \end{aligned} \tag{12}$$

where M is a rxc matrix that is the mean of the distribution, U is a rxr matrix and V is a cxc which are covariance of the rows and columns of the matrix respectively. In order to approximate matrix variate Gaussian posterior distributions, let $p_\theta(W), q_\phi(W)$ be a matrix variate Gaussian prior and posterior

distribution with parameters θ, ϕ and $p(x, y)$ is the distribution of training data. Then the lower bound on marginal likelihood is defined as:

$$\begin{aligned}\mathscr{L}(\phi;\theta) = E_{p(x,y)}[\log p(Y|X)] \leq \\ E_{p(x,y)}\left[\int q_\phi(W)\log\frac{p_\theta(W)p(Y|X,W)}{q_\phi(W)}dw\right] \\ = E_{p(x,y)}\Big[E_\phi(W)[\log p(Y|X,W)] \\ - KL(q_\phi(W)||p_\theta(W))\Big]\end{aligned} \tag{13}$$

This lower bound consists of an expected log-likelihood which is estimated with a Monte Carlo estimation and a KL-divergence which can be considered as complexity loss. For the simplicity of implementation, they approximate each of the covariances with a diagonal matrix, representing independence between rows and columns in the weight matrix. With this diagonal approximation to the covariance matrices, the KL-divergence between the matrix variate Gaussian posterior $q(W|M, \sigma_r^2 I, \sigma_c^2 I)$ and a matrix variate Gaussian prior $p(W|0, I, I)$ for a matrix of size rxc is:

$$\begin{aligned}KL(q(W|M,\sigma_r^2 I,\sigma_c^2 I)||p(W|0,I,I) = \\ \frac{1}{2}\Bigg(\Big(\sum_{i=1}^{r}\sigma_{r_i}^2\Big)\Big(\sum_{j=1}^{c}\sigma_{c_j}^2\Big) + ||M||_F^2 - rc \\ -c\Big(\sum_{i=1}^{r}\log\sigma_{r_i}^2\Big) - r\Big(\sum_{j=1}^{c}\log\sigma_{c_j}^2\Big)\Bigg)\end{aligned} \tag{14}$$

We use the implementation from [12] using a network with 3 layers and 150 hidden units per layer, with minibatch size 100 and dropout rate 0.25. We train for a hundred epochs, and use a hundred samples from the posterior for our analysis.

5.2 Generating Adversarial examples

5.2.1 Datasets

We used the cleverhans [14] library to generate adversarial examples in all cases. Two well known machine learning models and datasets were attacked: MNIST on a simple 2 layer CNN and CIFAR-10 on the VGG [20] network. The MNIST dataset was chosen as a baseline dataset, as much of the prior work has worked on this dataset. CIFAR-10 was chosen as an inexpensive coloured image dataset, one of our prescribed experimental objectives.

5.2.2 Generation

For MNIST, the default tensorflow MNIST dataset and CNN model (A two layer CNN with a first tensorflow 6 by 6 layer with stride 2 and another 5 by 5 layer with stride 1) were used. The basic CNN was trained for 6 epochs on 60000 examples, and the 10000 example test set was perturbed, using the trained model and the cleverhans API, to generate 10000 corresponding adversarial examples.

For CIFAR-10, we use the default Keras dataset, and load the VGG model with pre-trained weights [2]. We then also perform the same adversarial perturbations on the pre-trained model using cleverhans and the 10000 example test set.

We aimed to use a variety of input parameters, such that the perturbation amount accurately reflected adversaries that ranged from too strong, to too weak. For the FGSM method, we use ϵ values in the set [0.01, 0.03, 0.07, 0.1, 0.2, 0.3]. For the JSMA method, we use γ values in the set [0.01, 0.05, 0.1, 0.2, 0.3]. For both the FGSM ϵ value and the JMSA γ value, the default parameter value is 0.1.

Figure 1: Two CIFAR-10 FGSM adversarial images of a dog, on the left with ϵ=0.01 (lowest) and on the right with ϵ=0.3 (highest)

Figure 2: Two CIFAR-10 JSMA adversarial images of a dog, on the left with γ=0.01 (lowest) and on the right with γ=0.3 (highest)

For the baseline Gaussian noise measure, we also generated adversaries using

[2] https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3

Gaussian noise added to each colour channel of each image (or the single channel for MNIST). The noise added was sampled independently for each pixel and each colour channel from $\mathcal{N}(0, \sigma^2)$, for σ^2 values in the set: [0.05, 0.1, 0.15, 0.2, 0.25]. For any values that became negative or exceeded 1, the values were clipped to be 0 or 1 respectively.

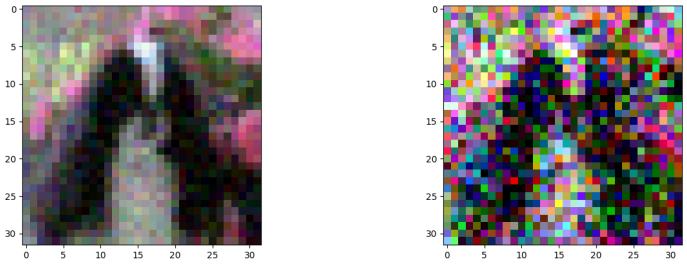


Figure 3: Two CIFAR-10 Gaussian perturbed images of a dog, on the left with σ^2=0.05 (lowest) and on the right with σ^2=0.25 (highest)

5.3 Measuring Uncertainty

We hypothesize that as we increase attack strength, model uncertainty about adversarial predictions will increase. We will compare the distribution of Bayesian uncertainty for adversarial samples to those of normal samples. For each type of attack, we will compute the model accuracy on the adversarial set and compare it with non-Bayesian neural network's accuracy, evaluating the effectiveness of attacks when BNN techniques are used. To evaluate model uncertainty in a classification problem, we use three different measures: Variation Ratio, Predicted Entropy and Mutual Information.

Variation Ratio To use variation ratio, we would sample a label from the output probabilities at the end of each stochastic forward pass for a test input x. Collecting a set of T labels y_t from multiple stochastic forward passes (generally known as Monte Carlo sampling) on the same input, we can then find the mode of the distribution $c^* = _{c=1,\dots,C} \sum_t \mathbb{I}[y^t = c]$, and the number of times it was sampled $f_x = \sum_t \mathbb{I}[y^t = c^*]$ where C is the number of possible labels for x. We have:

$$\text{variation ratio[x]} = 1 - \frac{f_x}{T} \tag{15}$$

Thus, the variation ratio is the relative number of times the majority class is not predicted and it shows how spread the distribution is around the mode [6]. It is 0 if the same class is predicted for all T samples and it reaches the maximum when every class is predicted uniformly. Thus, the higher the variation ratio, the higher the confusion of the model [18]. Despite its simplicity, this metric

is not quite reliable for measuring uncertainty in our classification task, since each sample from approximated distribution gives different probabilities to each class, but it is possible that the maximum probability always belongs to one class. Therefore all predictions would be the same as the mode, but we are not able to capture prediction variance with this metric.

Predicted Entropy Predicted Entropy $\mathbb{H}[y^*|x^*, D_{train}]$ is another measure that can refer to disorder or uncertainty. It reaches its maximum value when all classes are predicted to have equal uniform probability, and its minimum value of zero when one class probability is 1 and all others are 0. Predicted Entropy captures the average amount of information contained in the predictive distribution:

$$\mathbb{H}[y|x, D_{train}] = -\sum_c p(y = c|x, D_{train}) \log p(y = c|x, D_{train}) \tag{16}$$

like the variation ratio, we can approximate this by collecting the probabilities from T stochastic forward passes through the network [6].

Mutual Information Mutual information between the prediction y and the posterior over the model parameters w can reveal the model's confidence in its prediction. This measure of uncertainty is called Model Uncertainty as measured by Mutual Information (MUMMI) and is defined as:

$$\text{MUMMI} = \mathbb{H}[y|x, D_{train}] - \mathbb{E}_{p(W|D_{train})}[\mathbb{H}[y|x, w]] \tag{17}$$

Test points x that maximize the mutual information are points on which the model is uncertain on average, yet there exist model parameters that erroneously produce predictions with high confidence.
Mutual Information can be approximated with T samples from posterior distribution:

$$\begin{aligned}\mathbb{I}[y, w|x, D_{train}] = &-\sum_c \left(\frac{1}{T}\sum_t p(y = c|x, w_t)\right) \log \left(\frac{1}{T}\sum_t p(y = x|x, w_t)\right) \\ &+ \frac{1}{T}\sum_{c,t} p(y = c|x, w_t) \log p(y = c|x, w_t)\end{aligned} \tag{18}$$

To understand how mutual information works consider two set of probabilities for a classification task with two classes:

1. probability of each class is 0.5 (i.e. probability vectors are $\{(0.5, 0.5), ..., (0.5, 0.5)\}$)
2. for each class half of the probabilities equal to 0 and the other half equal to 1 (i.e. probability vectors are $\{(1, 0), (0, 1), ..., (1, 0)\}$)

The predictive entropy would be 0.5 for both examples, but the mutual information for the first example is 0 and for the second example is 0.5. In this case predictive entropy captures the *uncertainty in the prediction*, but mutual information captures the *model's confidence* in its output. [6]

6 Results

We trained five different Bayesian Neural Network model on MNIST dataset and we evaluate estimates of uncertainty for each of this model on three different type of adversarial examples. In our classification task, we consider both model's predictions probability and uncertainty for each example. Our expectation for this experiment is that, as the adversaial images become more noisy, the uncertainty increases and the model's confidence in class prediction will decrease as well.

In each plot in following tables, higher points density in bottom-right corner indicates high-certainty in class prediction and high-confidence in the prediction itself. For example in table 5 for BBB model, in left most column, the model has high confidence in its prediction. As adversarial perturbation becomes more intense in right most column, model's uncertainty increases significantly, which means BNN models can recognize adversarial attacks.

6.1 MNIST

Following tables are results of our experiments on 10000 adversarial images from MNIST test dataset.

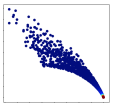
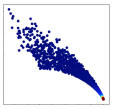
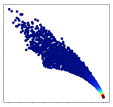
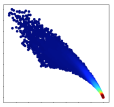
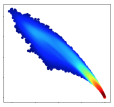
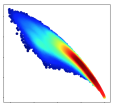
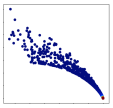
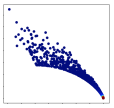
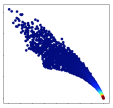
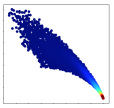
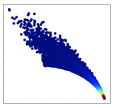
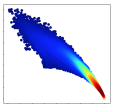
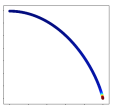
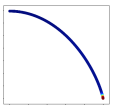
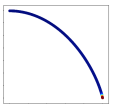
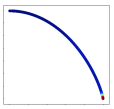
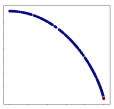
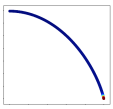
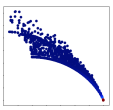
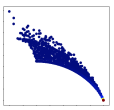
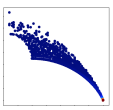
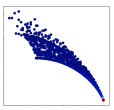
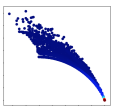
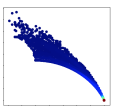
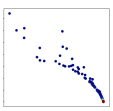
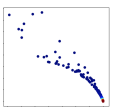
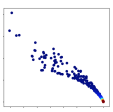
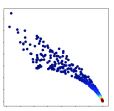
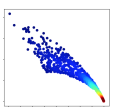
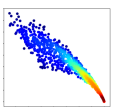
ϵ	0.01	0.03	0.07	0.1	0.2	0.3
						
BBB						
MC-Dropout						
PBP						
Pyro						
VMG						

Table 1: Scatter Plot of Probability of Predicted class (x-axis) vs. Predictive Entropy (y-axis) for MNIST dataset with **FGSM** adversarial data points. For each ϵ a representative sample is included. From red to blue the density decreases.

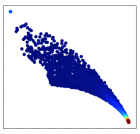
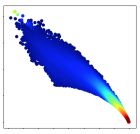
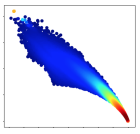
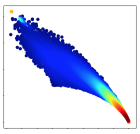
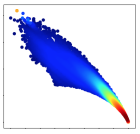
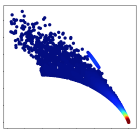
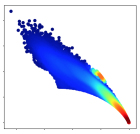
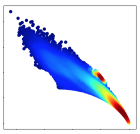
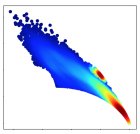
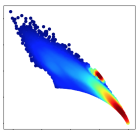
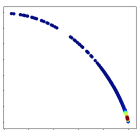
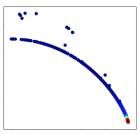
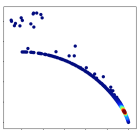
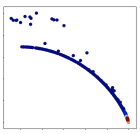
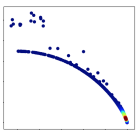
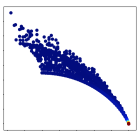
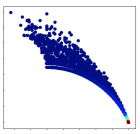
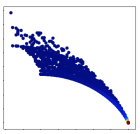
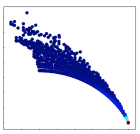
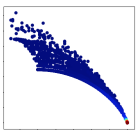
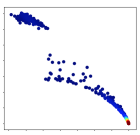
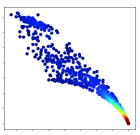
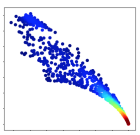
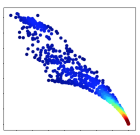
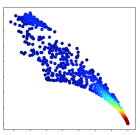
γ	0.01	0.05	0.1	0.2	0.3
					
BBB					
MC-Dropout					
PBP					
Pyro					
VMG					

Table 2: Scatter Plot of Probability of Predicted class (x-axis) vs. Predictive Entropy (y-axis) for MNIST dataset with **JSMA** adversarial data points. For each γ a representative sample is included. From red to blue the density decreases.

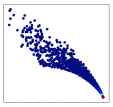
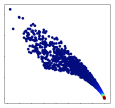
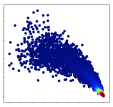
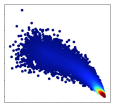
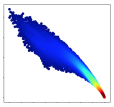
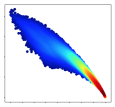
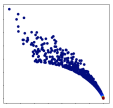
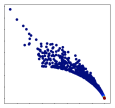
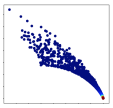
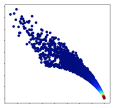
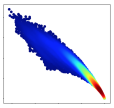
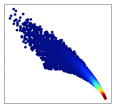
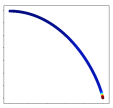
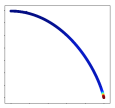
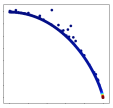
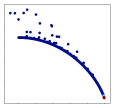
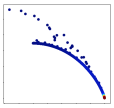
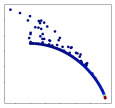
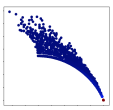
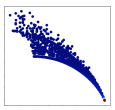
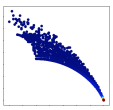
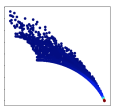
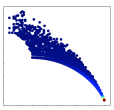
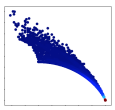
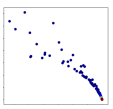
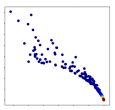
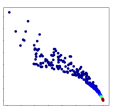
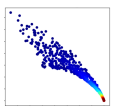
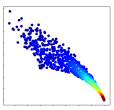
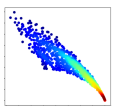
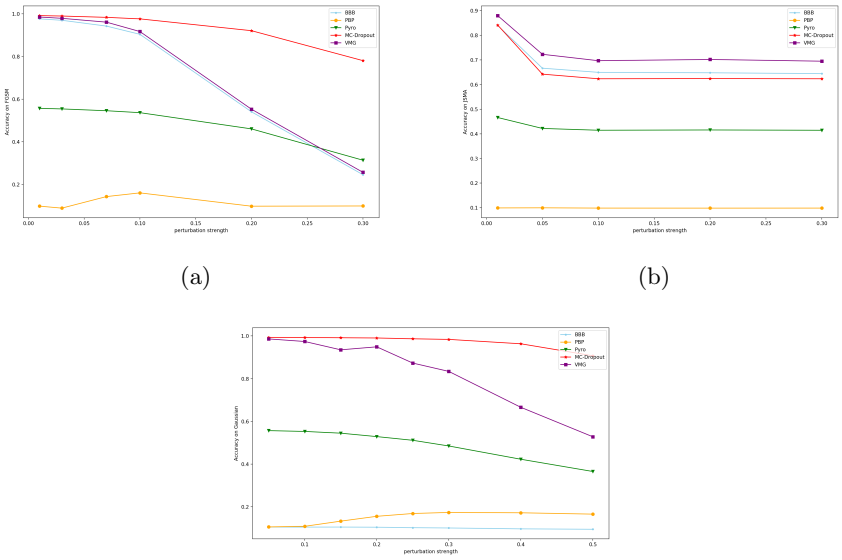
σ^2	0.05	0.1	0.2	0.3	0.4	0.5
						
BBB						
MC-Dropout						
PBP						
Pyro						
VMG						

Table 3: Scatter Plot of Probability of Predicted class (x-axis) vs. Predictive Entropy (y-axis) for MNIST dataset with **Gaussian** adversarial data points. For each σ^2 a representative sample is included. From red to blue the density decreases.



(a) (b)

(c)

Figure 4: plots of models' accuracy vs. perturbation intensity for (a) FGSM (b) GSMA (c) Gaussian perturbations on MNIST test examples. Comparing to other models, MC dropout has the most robustness to FGSM adversarial, while VMG and BBB have a significant drop in accuracy when perturbation increases. Although BBB can capture uncertainty better, MC-dropout is a more reliable model for prediction.

6.2 CIFAR

For performance reasons, we used VGG16 to generate a 512 output latent feature representation which we used for all of our BNN models except for MC Dropout which took in the raw pixel values.

Since VMG has the worst run time, we didn't evaluate VMG on CIFAR examples. According to results of our experiment, all models experinced significant drop in their accuracies, which indicates more complex models are needed to train coloured images. Also, the latent representation might have cause mistakes. Among four different models, it seems Pyro can capture uncertainties better than others.

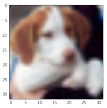
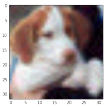
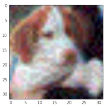
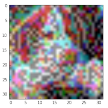
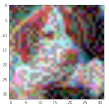
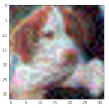
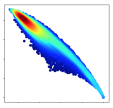
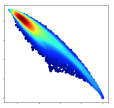
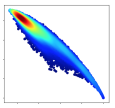
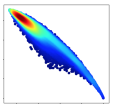
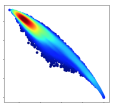
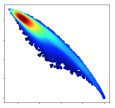
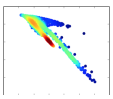
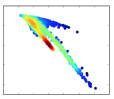
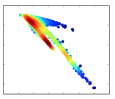
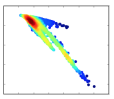
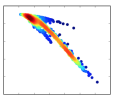
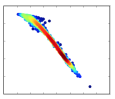
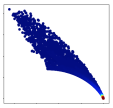
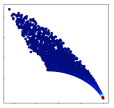
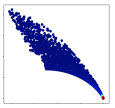
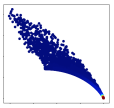
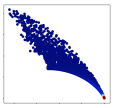
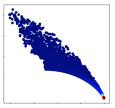
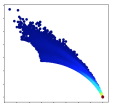
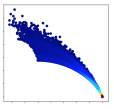
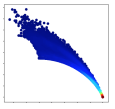
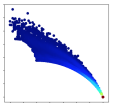
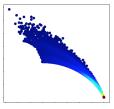
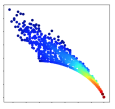
ϵ	0.01	0.03	0.07	0.1	0.2	0.3
						
BBB						
MC-Dropout						
PBP						
Pyro						

Table 4: Scatter Plot of Probability of Predicted class (x-axis) vs. Predictive Entropy (y-axis) for CIFAR10 dataset with **FGSM** adversarial data points. For each ϵ a representative sample is included. From red to blue the density decreases.

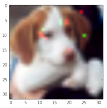
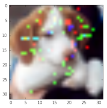
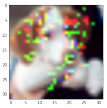
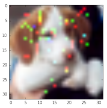
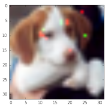
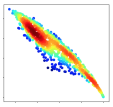
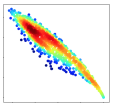
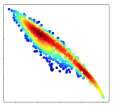
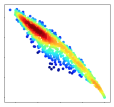
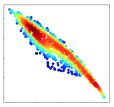
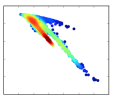
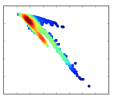
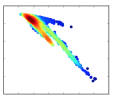
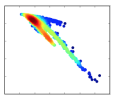
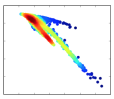
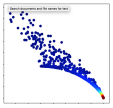
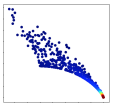
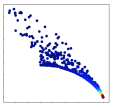
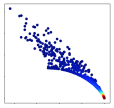
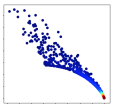
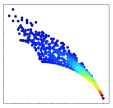
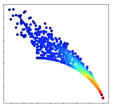
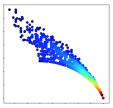
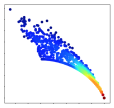
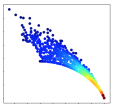
γ	0.01	0.05	0.1	0.2	0.3
					
BBB					
MC-Dropout					
PBP					
Pyro					

Table 5: Scatter Plot of Probability of Predicted class (x-axis) vs. Predictive Entropy (y-axis) for CIFAR10 dataset with **JSMA** adversarial data points. For each γ a representative sample is included. From red to blue the density decreases.

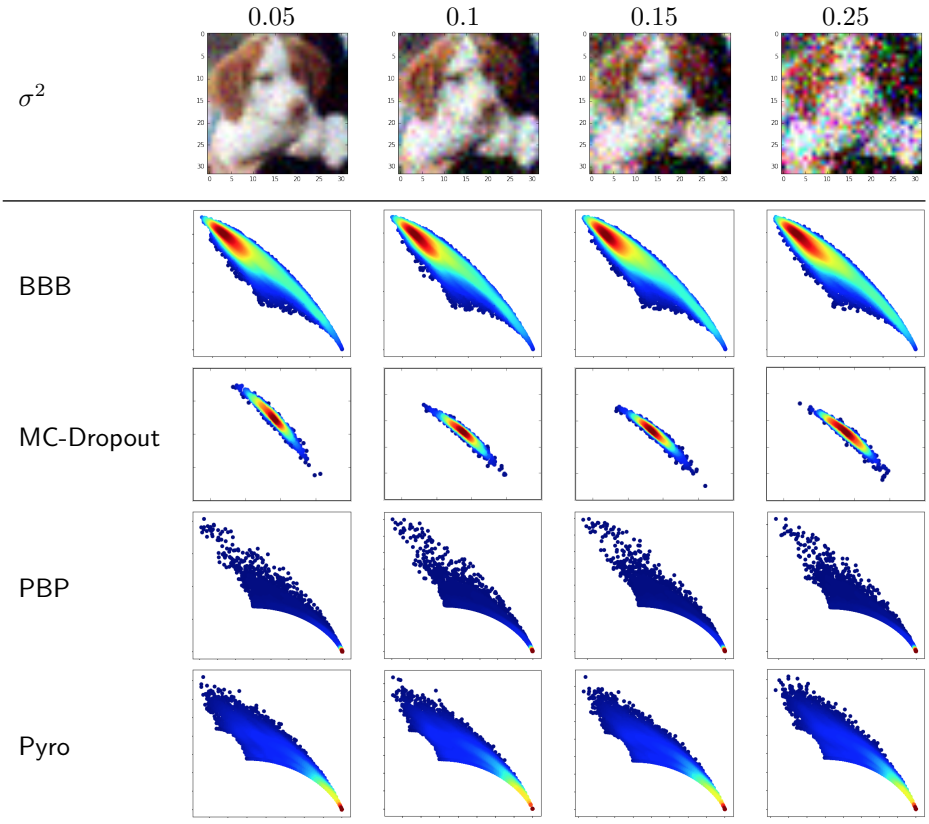
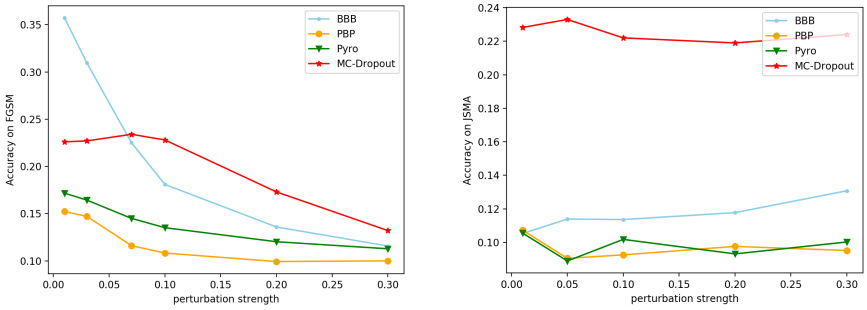
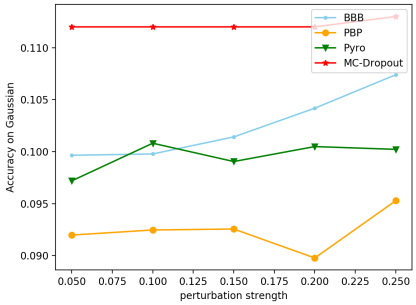


Table 6: Scatter Plot of Probability of Predicted class (x-axis) vs. Predictive Entropy (y-axis) for CIFAR dataset with **Gaussian** adversarial data points. For each σ^2 a representative sample is included. From red to blue the density decreases.



(a)

(b)



(c)

Figure 5: plots of models' accuracy vs. perturbation intensity for (a) FGSM (b) GSMA (c) Gaussian perturbations on CIFAR10 test dataset. Comparing to other models, MC dropout has the most robustness to FGSM adversarial,but MC-dropout can't capture uncertainty precisely.

7 Discussion

MNIST Empirically, our results showed that Bayes by backpropagation (BBB) and the VMG are the most effective at capturing uncertainty with increasing levels of attack by our different adversarial images. This suggests that the variance on the weight distributions of the two are the most spread, capturing the potential uncertainty when presented adversary images. Interestingly, this does not necessarily correspond to having the best accuracy when face with adversarial images as in two out of our three adversaries we see that MC dropout seems to shows the best preservation in terms of accuracy with increasing levels of adversary.

Our Pyro model seems unchanged with any degree of attack, maintaining the same level of accuracy despite an increase of attack strength. This is also demonstrated in the same level of predictive entropy where we saw no shift in the

density of accuracy for our adversaries. This suggested that perhaps for more robust classification the Pyro API provides an effective variational approach against such attacks.

Our PBP results seem to perform the worst, showing almost no shift in density of predictive entropy with increasing levels of adversarial attack. Given that little variance is shown in the plots, it seems our approach to try and simulate a classification version of the approach with regression was ineffective. In a strictly classification setting, the model showed to work well, achieving 90% accuracy on the unaltered images, but plummeting in performance with even the smallest of perturbations. These seem to suggest that PBP is highly ineffective in dealing with even the smallest of changes to the trained input.

CIFAR Our results when generalizing to colored images are mixed. On one hand, we are better at capturing uncertainty for several of our Bayesian approaches as attack strength increases. However, the effectiveness as classifiers is not as good as MNIST, this may be due to hyperparameter configurations for each of our Bayesian methods, as well as the use of latent feature representations of the images instead of the original pixel values. We performed analysis using the model configurations we tuned for the use on MNIST, which doesn't necessarily mean they would transfer well. Further work is required to definitively answer the utility of BNNs in the colored image domain.

8 Conclusion

We showed new findings over the current state of the art regarding BNNs and their robustness in modelling the uncertainty when predicting adversarial examples. For all attack types, as strength increases we observe an increased measurement of uncertainty for BBB, MC dropout, and VMG. More tuning is required for colored images due to the increased complexity of hyperparameter selection.

References

[1] Pyro deep universal probabalistic programming. http://pyro.ai/. Accessed: 2018-02-23.

[2] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[3] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, 2017.

[4] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, 2009.

[5] 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*, 2017.

[6] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

[7] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

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

[9] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[10] José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1861–1869. JMLR.org, 2015.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1708–1716. JMLR.org, 2016.

[13] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[14] Ian Goodfellow Reuben Feinman Fartash Faghri Alexander Matyasko Karen Hambardzumyan Yi-Lin Juang Alexey Kurakin Ryan Sheatsley Abhibhav Garg Yen-Chen Lin Nicolas Papernot, Nicholas Carlini. cleverhans v2.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2017.

[15] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy*, EuroS&P, 2016.

[16] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, 2017.

[17] Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 814–822, Reykjavik, Iceland, 22–25 Apr 2014. PMLR.

[18] Ambrish Rawat, Martin Wistuba, and Maria-Irina Nicolae. Adversarial phenomenon in the eyes of bayesian deep learning. *arXiv preprint arXiv:1711.08244*, 2017.

[19] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3528–3536. Curran Associates, Inc., 2015.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014.

[21] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *CoRR*, abs/1710.08864, 2017.

[22] David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *CoRR*, abs/1301.1299, 2013.

[23] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *arXiv preprint arXiv:1712.07107*, 2017.