# Modelling Uncertainty in Deep Learning

Ninad Khargonkar
University of Massachusetts, Amherst
nkhargonkar@umass.edu

## Abstract

*Deep learning based models are having tremendous success in domains like computer vision and natural language processing. However there is still a want for clear understanding of how and what the model actually learn, and making the models more interpretable. We look into estimating uncertainty measures from the output of a deep neural network in the classification setting using test time monte-carlo dropout sampling as a way to approximate from the true posterior over the weights of neural network. Further on, we perform anomaly detection through a linear classifier to evaluate the quality of the uncertainty estimates by using these estimates as features to detect images not seen in the training set.*

## 1. Introduction

Currently most of the deep learning models are treated as black-boxes for performing various tasks. However with the growth of artificial intelligence systems in real-world applications, there is an increasing need for interpretability of the models and their outputs especially in the face of ambiguous inputs (Fig.1). Bayesian deep learning is one approach in which the models make use of bayesian modeling to answer questions like what does the model actually learn and why does it work for a particular problem.

In this project we intend to go into how to obtain uncertainty estimates using approximate methods for the full bayesian posterior inference and also how to evaluate the quality of the estimates using anomaly detection as proxy task. Initially we will look what types of uncertainties we are dealing with and how to incorporate them into our models. We will look into how the (heuristic/guideline) of using dropout while training deep neural networks can actually be used for approximate bayesian inference. Then we will use the uncertainty estimates from our model and use them as features in an anomaly detection task for finding out images which were out of the training set.

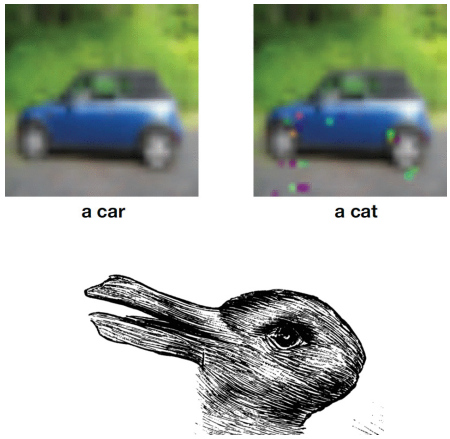


Figure 1. The top image (from [8]) shows that that a neural network can be fooled by addition of some noise. The bottom image is an example on inherently ambiguous image. An image classification system can benefit from uncertainty estimates for such examples in the overall model pipeline.

We will mainly focus on the task of image classification on the CIFAR-10 and MNIST data sets due to their simple nature and relative ease in trying new experiments on them. Our goal here is not to achieve a state of the art performance on classification accuracy, but instead we would like to see how well the various metrics used for incorporating uncertainty into the model are working, especially with some augmented/transformed data. Getting a good uncertainty estimate is not a particularly well defined task, at least for classification problems since the model output is not a single value but a probability vector over all the classes.

The uncertainty estimate can be split into two broad categories: epistemic uncertainty and aleatoric uncertainty. Epistemic uncertainty focuses on what our model is not able to capture through the training data. Basically, it signifies the ignorance on data it has never seen before, mostly as a consequence of the fact that the model did not encounter a particular kind of data during training. Therefore it can be also referred to as the model uncertainty and it usually decreases once the model is provided with

more amount of training data. High epistemic uncertainty can alert us about novel/unknown situations for the model and this is also important for models who are trained on a small amount of training data.

On the other hand aleatoric uncertainty deals with information which the input data is not able to clearly explain/convey to the model i.e what we can't understand/extract from the input. For example, in the case of images it may arise due to occlusions which may interfere with related semantic tasks. Basically, it can be thought of as a 'sensor' uncertainty where an ideal 'sensor' would be able to provide all required information with a high level of precision. It can be further divided into two categories: 'heteroscedastic' and 'homoscedastic'. Heteroscedastic uncertainty depends on the model input (as a deterministic function) and therefore can be leveraged for real time applications. Homoscedastic stays constant across the inputs and is a property of the task being performed.

## 2. Related Work

The survey papers by Wang et al [10] and Polson et al [9] serve as a general introduction to bayesian deep learning and its applications in various domains. The paper by Gal et al [3] and his PhD thesis [1] lay out the theoretical framework of using test time monte carlo dropout to approximate the posterior distribution over the weights. Also in [2] they go into specifics of such approximation in convolutional neural network.
Kendall et al [6] go into the motivation and formulation for uncertainties described above along with showing results on regression based computer vision tasks like segmentation and depth prediction. Oliveira et al [7] provide a simple guideline to test the quality of uncertainty estimates that we obtain using anomaly detection on out of training set images.

## 3. Approach

### 3.1. Epistemic Uncertainty

Modeling epistemic uncertainty requires us to put distributions on the model parameters which can be a challenge we dealing with large networks. A really interesting result shown in [3] is that approximate Bayesian inference (in a deep Gaussian Process) can be modelled relatively well using Monte Carlo Droput sampling (by putting a Bernoulli distribution on the network parameters). As full Bayesian inference is usually expensive for a given test input (due to posterior computation), so variational inference is used and therefore for the test input, we basically sample from weights from the variational distribution (here Bernoulli) and take the expected value. Here, $p(y^*|x^*, D)$ is replaced by its approximate predictive counterpart $q_\theta(y^*|x^*, D)$ which is defined to be the (approximate) predictive distribution with variational distribution $q_\theta(W)$ being used instead of the true posterior for $W$ (Here $D$ is the data the model has already seen). Therefore, finally at test time we have:

$$q_\theta(y^*|x^*, D) \approx \frac{1}{N}\sum_{n=1}^{N} p(y^*|x^*, W_t)$$

What this basically means is that in practice, at test time, we randomly sample the weights using dropout masks $W_t \sim q_\theta(W)$ to get a prediction and perform this sampling multiple times to create a predictive distribution for a single test input $x^*$. We can then use its variance (for regression tasks) or the entropy of the probability vector (for classification tasks) of this distribution as a reflection of the model's epistemic uncertainty.

### 3.2. Aleatoric uncertainty

Since aleatoric uncertainty is a function of the input data, we can incorporate it into the loss function. The aleatoric uncertainty (heteroscedastic version) can be modelled by tweaking the model to also predict the input variance $\sigma^2$ and changing the loss function to include the input variance (such that for a very wrong prediction, the uncertainty goes up). Here the model does not have training data to learn the variance $\sigma^2$ and it is implicitly learned from the loss function.

The loss function for a classification task is a modified version of the cross-entropy loss. Here the input variance $\sigma_i^W$ (for test example $x_i$ in model $W$) is induced by corrupting the logit values $f_i^W$ before the softmax layer by Gaussian noise as described in [6]. The Gaussian noise has a mean of zero and $\sigma^2 =$ predicted variance.

$$\hat{x_i} = f_i^W + \sigma_i^W \epsilon_t,$$

$$\epsilon_t \sim \mathcal{N}(0, I)$$

Therefore, we are predicting a variance for each of the logit value and hence the uncertainty is calculated in the logit space. After the distortion, the cross entropy loss is calculated using the corrupted values. To approximate integration of the Gaussian out of the log likelihood, we take Monte Carlo samples and then take average of the samples for the loss. This is a relatively fast step since the logits have already been calculated as opposed to the case for epistemic uncertainty where we had to make multiple passes through the network to get the samples.

### 3.3. Uncertainty evaluation

The proxy task of anomaly detection can be used to test the quality of uncertainty (epistemic) measures that we

obtain using the using the approximate method described above. We will train a simple linear logistic regression model on top of some features which are basically the uncertainty measures. The task will be to identify which of the test sample images were not seen in training and this will probably also give an insight into the classification confidence of the model. A representative model diagram is shown in Fig.2

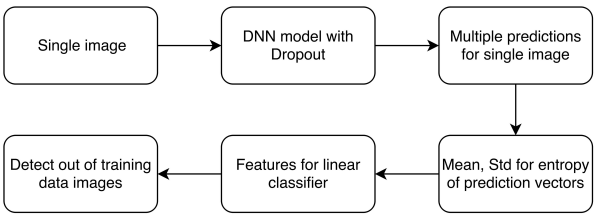


Figure 2. The top image (from [8]) shows that that a neural network can be fooled by addition of some noise. The bottom image is an example on inherently ambiguous image. An image classification system can benefit from uncertainty estimates for such examples in the overall model pipeline.

To build such a system, we train the neural network in the usual fashion but with certain classes excluded during training. The classes which were not included during training will act as ground truth 'anomalies' during training of the linear classifier. The performance of linear classifier will be evaluated using the standard metric of calculating the area under the curve under the precision recall curve. A higher value for the area under the curve will signify that the uncertainty estimates are useful in distinguishing out of training set images.

## 4. Experiment

I trained a relatively simple CNN on CIFAR-10 to get the base model and then tweaked its loss functions to model the uncertainties. My base model had the architecture with softmax layer at the end: [[Conv-Relu-Pool-Dropout] x 2 – Affine – Droput – Affine]
For MNIST, I used fully connected neural network with two hidden layers having 512 and 256 neurons respectively. In both the models, the dropout probability after the fully connected layers was 0.5 while in the CNN for CIFAR-10 after the convolutional layers, the dropout probability was 0.25.

The cifar-10 model was able to achieve about 80% accuracy after training for 40 epochs while the mnist fully connected model got around 95% accuracy on the test set. As stated earlier, the main goal of my experiment is not to achieve a state of the art classification accuracy but to experiment with the uncertainties for which this relatively simple to train (and tweak) model was sufficient. In figures 3 and 4, a sample image for MNIST and its occluded counterpart with their respective uncertainty measures are shown. It can be seen that the uncertainty in prediction increases as we start providing less information to the model which is expected.

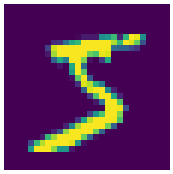

Figure 3. Original: 0.243

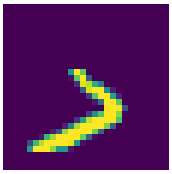

Figure 4. Modified: 0.325

Some reference code for turning on test time dropout in keras is shown below:

```
'''
Example code to demonstrate test time dropout
in keras using a custom layer
'''
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers.core import Lambda
from keras import backend as K

model = Sequential()
model.add(Dense(512, input_shape=(28,28)))
# use this:
model.add(Lambda(lambda x: K.dropout(x, level =
    0.5)))
#instead of:
model.add(Dropout(0.5))
```

dropout.py

### 4.1. Augmented images

The epistemic and aleatoric uncertainties for cifar-10 were calculated on a random subset of 1000 examples from the test set. Furthermore, I applied gamma augmentation to test the response for the uncertainties and see how their value change on seeing an augmented image. For one set I used a gamma value of 0.6 (decreasing the pixel intensity) and for another set I used a gamma value of 2.1 (increasing the pixel intensity). The mean and standard deviations for the uncertainties are provided in the tables 1 and 2.

| Images | Mean | Std |
|---|---|---|
| Original | 0.394 | 0.473 |
| Gamma=0.6 | 0.432 | 0.492 |
| Gamma=2.1 | 0.506 | 0.504 |

Table 1. Epistemic uncertainty

As we can see from the above tables, the values for aleatoric uncertainty are very low for all three types of images when compared to the epistemic uncertainty. I intend to analyze further as to why is it turning to be so low.

| Images | Mean | Std |
|---|---|---|
| Original | 1.38e-07 | 9.43e-07 |
| Gamma=0.6 | 2.43e-07 | 2.52e-06 |
| Gamma=2.1 | 4.90e-07 | 2.81e-06 |

Table 2. Aleatoric uncertainty

One possible reason (apart from some bug in the code!) perhaps could be that since this is a classification task, most of the images (in the training and test set) are by design 'clean' and do not contain inherent noise like occlusions or low contrast.

Also, a change that is noticed in both the tables is that the value for uncertainty increases for the gamma transformed images. This is expected since the model was trained for gamma=1.0 images and brightness/darknes could be interpreted as noise in the image and it also supported by the fact that the accuracies for both of these transformed sets fall down by a value of 10%. The images who have maximum aleatoric and epistemic uncertainty are shown in the figures 5 and 6 respectively.

### 4.2. Test set results

The class labels for Fig.5 (going in a clockwise manner) are deer, frog, plane and ship and for Fig.6 they are deer, plane, ship and truck. In fig.1 we can observe that the image data is somwhat not clear enough while in fig.2, the images are far apart from the usual representative images of their corresponding class.

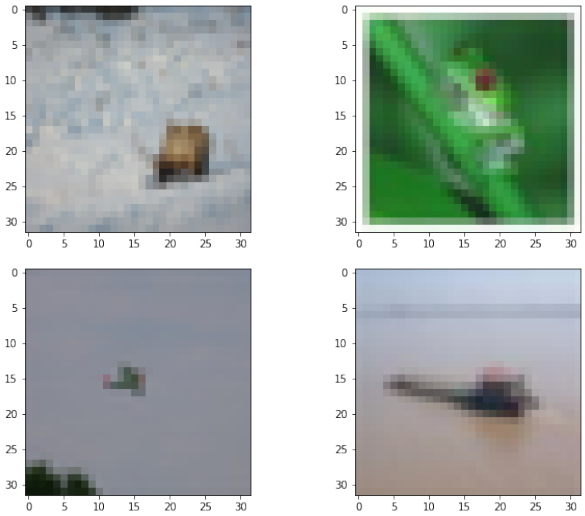

Figure 5. Highest values for aleatoric uncertainty

Figure 6. Highest values for epistemic uncertainty

### 4.3. Anomaly detection

For the anomaly detection task, we exclude the last three classes in both mnist (the digits 7,8 and 9) and cifar-10 (the categories horse, ship and truck) from the deep neural network training phase and hence in essence they are the anomalies for the linear classifier. We then train two classifiers, one has the uncertainty estimates of the images as extra features while training while the other one only has access to the single entropy of probability vector as a feature (the model without access to uncertainty estimates).

For a single test image when we perform test time monte-carlo dropout, we get multiple predictions(probability vectors). From these we take the entropy of the mean probability vector, mean of entropy values across predictions and the variance of the entropy values across predictions as our refined features to the anomaly detector.

| Features | Score |
|---|---|
| w/o Uncertainty | 0.69 |
| with Uncertainty | 0.715 |

Table 3. Area under the curve score for a anomaly classifier trained on mnist datausing both the refined features (which use uncertainty estimates) and just the entropy of final prediction vector.

The results of the classifier for mnist data are shown in table 3. It can be seen that using the extra uncertainty information as a feature helped in improving the performance of the classifier in detecting out of training set examples. Unfortunately we were not able to get such clear

demarcation in perfromance for cifar-10 data set possibly due lack of clear cut distinctions in the respective class images itself.

While a neural network might never see a horse image in training, it can be probably infer that is some kind of animal and not a vehicle and thus the extra information from the uncertainty is not that useful here. In order to use the anomaly detection proxy task as a way to evaluate the quality of uncertainty measure, it may probably need a clear boundary 'in' (training) and 'out' (not in training) images for the neural network training phase in order to truly test the effectiveness of such measures.

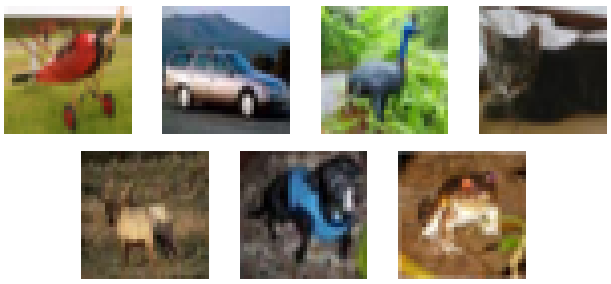

Figure 7. 'In' classes in training the convnet

Figure 8. 'Out' classes in training the convnet (anomalies)

## 5. Conclusion

Although we are able to get results which are in line with our expectations from theory, there are some pressing points which may need to be addressed. The extremely low values for aleatoric uncertainty indicate that either the data on which it was tested on does not show enough variation to induce a high value or the formulation for the classification setting needs to be tweaked. Another possible bottleneck could be the relatively large running time for the epistemic uncertainty calculations which stems from multiple forward passes through the network. Getting around this could help in incorporating it into real-time systems and applications. As for the anomaly detection task, one could also look in to other ways for tuning the dropout probability as shown in [4]. Further natural extensions of these results could be on more complex data sets and also in using the uncertainty information in adaptive learning settings [5] or querying which training examples to ask for.

## References


[1] Y. Gal. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[2] Y. Gal and Z. Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

[3] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning*, pages 1050–1059, 2016.

[4] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. *arXiv preprint arXiv:1705.07832*, 2017.

[5] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

[6] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *arXiv preprint arXiv:1703.04977*, 2017.

[7] R. Oliveira, P. Tabacof, and E. Valle. Known unknowns: Uncertainty quality in bayesian neural networks. *arXiv preprint arXiv:1612.01251*, 2016.

[8] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.

[9] N. Polson and V. Sokolov. Deep learning: A bayesian perspective. *arXiv preprint arXiv:1706.00473*, 2017.

[10] H. Wang and D.-Y. Yeung. Towards bayesian deep learning: A survey. *arXiv preprint arXiv:1604.01662*, 2016.

# A. Supplementary material

## A.1. Short visualization video

The first thing that might come to our mind when talking about using the entropy of probability vector as the uncertainty estimate is why not just take a single prediction and use the single value of entropy. One possible drawback with this is that even if our model predicts a certain class with a very high confidence (i.e low value for the entropy), it might so happen that the image was occluded and the model thought of it as something entirely different. This is not always an ideal situation especially in real world systems!

We can see this happening in this youtube video `https://youtu.be/th1Ve-9bZnQ`. Basically I progressively set the first i-rows (i goes from 1 to 28) of a sample mnist image (digit 5) to zero and calculated the entropy (uncertainty estimate) of the single prediction vector. One can observe that the uncertainty is not strictly increasing and there are dips in between perhaps from the model thinking that it some different digit like 7 or 1.

## A.2. Moodle submission

I have also included the image frames and the video in the moodle submission of supplementary material. In addition I also included some code scripts that I used in the project, however I did not include the trained models and extracted features in the submission since the file size would exceed 10MB.