Implementation and analysis of Deep Bayesian Active Learning with Image Data

Anonymous CVPR submission

Paper ID ****

Abstract

Active Learning is a branch of machine learning that tries to automate the process of dataset generation. However, the main problem with active learning was that this method was not reliable. This was because of improper uncertainty estimation methods.This was changed after the invention of dropout as a Bayesian approximation method. Using dropout we can estimate the uncertainty of neural networks. Thus, extrapolating this method Gal et al.[2] proposed a uncertainty based active learning method in [2]. In this paper I am going to reimplement the method implemented in [2] and I am going to present my analysis of the same

1. Introduction

Active Learning is a branch of machine learning that tries to deal with automatic annotation of dataset. Currently, the only way to annotate a dataset is by manually hiring an annotator and training them to annotate a dataset. However, this process is time consuming and expensive in nature. Hence there has been a need to create machine learning models that could automate the process of dataset annotation. This process is known as active learning.

As mentioned above, the entire procedure of dataset annotation is known as active learning. However, current machine/deep learning models are not very efficient in active learning[2]. As mentioned in [2] most of the active learning methods have been focused on low dimensional data.

One main reason for not scaling the deep learning models was the lack of uncertainity estimation[2]. It is a well known fact that deep learning models are overconfident on out of distribution datasets[1]. Hence when these deep learning models give their prediction, there is a high chance that they generate a wrong prediction with high "confidence". This would be a disastrous situation in active learning as a deep learning model would be re-trained on a wrongly annotated dataset. This would make the model inaccurate and would result in poor prediction results.

Hence for active learning to work we need to estimate the uncertainity of deep learning models. The uncertainity metric would give us an estimate on how uncertain our model is on the given prediction. This would give us a "second opinion" on our model thereby preventing us from feeding wrongly annotated labels to the deep learning model.

In this paper I would try to reimplement the method introduced by Gal et al. in [2]. I would try to first implement the uncertainity estimation of a deep learning model by introducing dropout in the model[1]. Post the estimation of uncertainity I would try to implement the methodology mentioned in [2].

2. Steps in Active Learning

In the previous section I mentioned that active learning coud automate the process of dataset annotation. Therefore, it is necessary to explain the entire pipeline of active learning before we proceed with the implementation.Fig 1gives an overview of the pipeline used in active learning. The following are the steps involved in active learning:

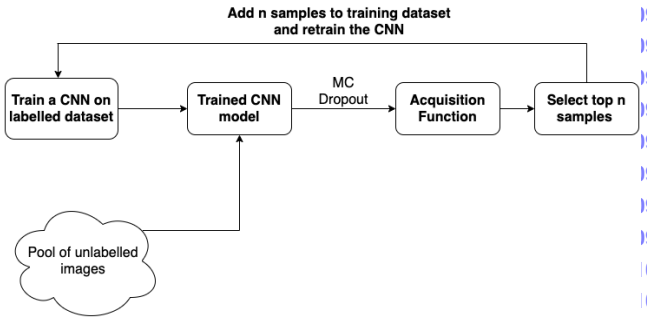

Figure 1. Active Learning pipeline

1. **Training of deep learning model:** This is the first step in active learning. We first take a dataset and we train our deep learning model on the dataset.

2. **Uncerainity estimation for pool set:** In this step we take the model trained in previous stage and perform dropout based uncertainity estimation[1]. This is done by passing the same pool dataset multiple times through the dropout induced neural network. On doing a MC approximation we can then get average probability for every sample in pool dataset. The readers can refer [1] for further details about uncertainity estimation.

3. **Passing the average probability values to acquisition function(AF):** The acquisition functions help in estimating the uncertainity in the data obtained from previous step. The different acquisition functions will be mentioned in Section 3

Once we have the values from the acquisition function we select the pool values which have the maximum AF values. These values are added to the training dataset as shown in figure 1 and the model mentioned in step 1 is retrained using the existing and the new dataset. Thus in a nutshell active learning is the retraining of the neural network by automatically annotating the dataset.

3. Theory

In the previous section I mentioned the pipeline followed in active learning. In this section I will give a theorotical explanation of how each component works.

1. **Training of a neural network:** This step involves training a standard Deterministic Neural Network. In this paper I have trained a Convolutional Neural Network(CNN)[6]. Due to limited amount of space I would not go through the training of deterministic CNN. The readers can refer [5] for a detailed explanation on how a CNN works.

2. **Uncertainty estimation in Deterministic Neural Networks:** As mentioned in [1] deterministic NN are overconfident models. Hence [1] had proposed a way of converting the deterministic models into a Bayesian Neural Network(BNN). The Deterministic models could be converted to BNN by adding dropout at test time[7, 2, 3] and then performing MC approximation on them[1].

3. **Acquisition Function:**The Acquisition Function is responsible for estimating the uncertainty of the samples[2]. According to the output of the Acquisition Function we can determine if the samples can be used for retraining the neural network or not. The uncertainty can be estimated using multiple different functions namely:

    - **Entropy:** This is one of the most common and simple ways of describing uncertainty. Entropy can be calculated using the function mentioned below

$$H \approx -\sum_c (\frac{1}{T}\sum_t p_c^t) Log(\frac{1}{T}\sum_t p_c^t)$$

Figure 2. Entropy formula. Referred from here

    - **Var Ratio:** Variational Ratio is another way of estimating uncertainty. It can be estimated using the function mentioned below

$$\text{variation-ratio}[\mathbf{x}] := 1 - \max_y p(y|x, \mathcal{D}_{\text{train}})$$

Figure 3. Variational Ratio formula. Referred from[2]

    - **Bayesian Active Learning by Disagreement(BALD):** This is another method of calculating uncertainity. In this method we try to minimise the samples on which we arent confident about. Thus we only pick those samples which we are confident about.

$$I(y; \omega|x, D_{train}) \approx -\sum_c (\frac{1}{T}\sum_t p_c^t) Log(\frac{1}{T}\sum_t p_c^t) + \frac{1}{T}\sum_{t,c} p_c^t Log p_c^t$$

Figure 4. BALD formula. Referred from here

4. Methodology

In this section I will explain the methodology that I have followed to implement the paper. As mentioned in [2], I have used the 2 mentioned dataset namely: MNIST and ISIC [2]. In addition to the two datasets, I have also tested the method on CIFAR-10 dataset. The CIFAR and MNIST are prebuilt in most of the libraries hence they have not been cited in this paper. In order to test the methodology I have used the architecture mentioned in Fig 5 It is to be noted that I have not used the architecture mentioned in [2]. I have used a shallower CNN because of limited GPU resources.

The following steps are followed for implementing the methodology mentioned in [2]:

1. Take the train dataset and split it into train and pool dataset. The train dataset contains very low number of samples while the pool dataset contains most of the samples. Keep the test dataset aside and it will be used only for testing. For all the datasets I have initially trained the CNN on 20 samples and kept the remaining amount as pool dataset. I have chosen a fewer samples for training due to limited GPU computation power.

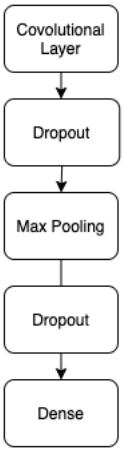

Figure 5. Architecture of CNN

2. Train the CNN mentioned above using the training samples.

3. Now take the pool dataset and pass the pool dataset multiple times through the trained CNN. For my implementation I have passed it just 3 times due to limited CPU/GPU resources.**Note that the dropout should be activated during each of these passes.** The softmax layer at the output of CNN would give us probability based predictions for every class. Take the average of these predictions for every sample. This step is known as MC approximation as mentioned in Section 3(2nd point).

4. Pass the MC Approximated outputs through the acquisition functions mentioned above. Pick the top 10 samples along with their associated labels and add them to the training pool which was mentioned in step 1.

5. Repeat the above steps multiple times. In my implementation I have repeated this step 50 times(as mentioned in [2])

The above steps have been used to test the 3 datasets which I have mentioned above.

5. Results

In this section I will compare the results of my implementation with the results mentioned in [2]. For ease of reading I have divided this section in 3 subsections based on the dataset in consideration.

5.1. MNIST

As mentioned in Section 3, I have used 3 acquisition functions namely: BALD, Entropy and Variational Ratios. Figure 6 gives a comparative analysis of the results obtained using my implementation and the author's implementation.

As visible in Fig 6 I have obtained an accuracy of roughly 90% for all the acquisition functions. This accuracy is very similar to the accuracy obtained by the author. The author has obtained higher accuracy due to a deeper CNN. This code was trained using Colab GPU and is written by me except the preprocessing sections. I have added the necessary citations in the code file.

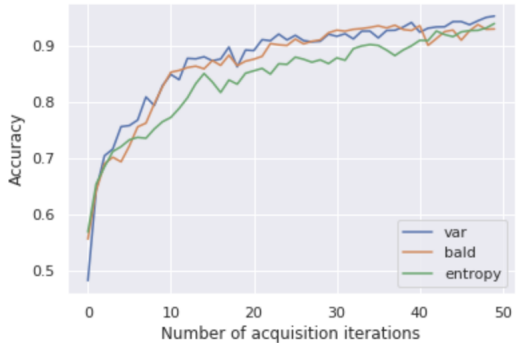

Figure 6. Results after running the MNIST code

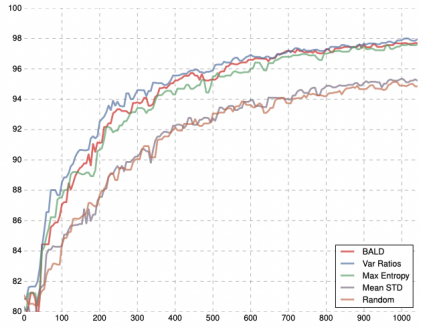

Figure 7. Results after running the MNIST code

5.2. ISIC:

In order to test this dataset the author has used AUC and also passed the test set multiple times through the model[2]. However, I did not have time to implement the AUC function and follow the author's procedure mentioned in [2].

Hence I have performed MC dropout on the dataset and have calculated the accuracy instead of AUC. This method is very similar to the way MNIST dataset was tested in this paper. The only difference is that I have just run 4 acquisition iterations instead of the standard 50. This was done because the dataset is heavily imbalanced and I feared that the model might overfit and give wrong results.

Since my method of testing was not similar to author's testing method, I have not done a comparative analysis for this dataset. The output for my code is as shown in Fig 8

This code is completely written by me and the code is run on a dedicated CPU having 16GB ram and a GPU of 16GB memory.

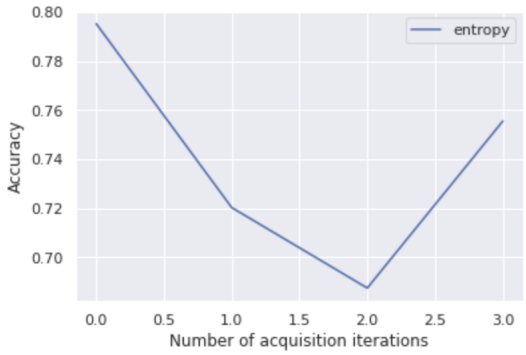

Figure 8. Results obtained after running ISIC code

5.3. CIFAR10:

5.3.1 Results on my architecture

This implementation is very similar to the MNIST implementation. Since Gal et al.[2] has not tested this method on CIFAR10, I have just provided the results of my code. The accuracy of my model on Cifar is as shown in Fig 9. Since I had obtained a very low accuracy on CIFAR10 I thought that it would be appropriate to test this method on a better model to see if I still get a poor accuracy. Hence in the next section I will discuss the training of CIFAR on a resnet model.Similar to ISIC dataset, this code was run on on a dedicated CPU having 16GB ram and a GPU of 16Gb memory. This code has been written by me except the preprocessing section. I have cited the appropriate references in the code file.

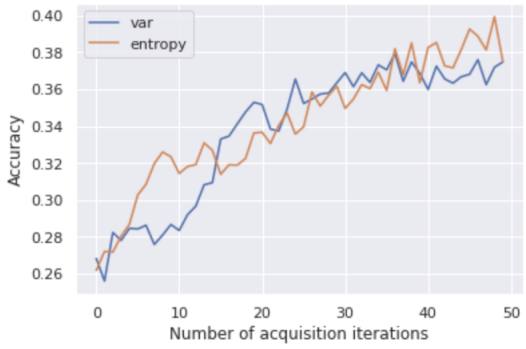

Figure 9. Results obtained after running my code

5.3.2 Results on Resnet:

As mentioned in figure 9, the accuracy obtained using my code was very low for CIFAR10 dataset. Hence I decided to test this method on another architecture. My initial guess was that this method did not work on CIFAR10 because the CNN architecture mentioned in Table 5 is very shallow for CIFAR10. Hence I decided to test this method on a well established model named ResNet. It is to be noted that I have **not** written this implementation. I have directly picked up the from this repository. The results after running the code are mentioned in Fig 10.

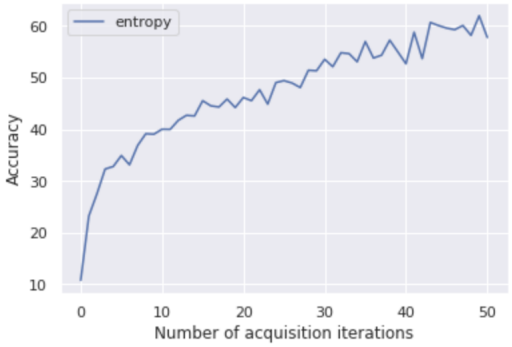

Figure 10. Results obtained after running the code from here

6. Analysis and Discussion:

In the previous section I have performed a comparative analysis of my implementation with the author's implementation. In this section I will analyse the results.

1. **Poor Accuracy on CIFAR10:** On analysing the results of CIFAR10 using both the methods I observed that the results are not very impressive even after using a better model. I believe that this could be because the MC-dropout method [1] gets stuck in a local optima[2]. The author in [2] has mentioned that we can tackle this problem by reinitialising the model after every MC-dropout iteration. However, I have not tried that in this paper. Hence that could possibly be the result of poor accuracy on CIFAR10

2. **Large amount of computational resources required:**On implementing this method I realised that this method requires a large amount of computational resources. The important thing to notice here is that this method requires more CPU power than GPU power, as it takes a large amount of time to comb through thousands of datapoints in pool set for each acquisition step. This was one of the primary reason I decided not to implement the ensemble method[4] for uncertainity estimation.

3. **Adding dropout after every layer has an effect on accuracy:** In order for this method to work the CNN should give high accuracy on the training dataset. However, while implementing the code, I noticed that

if dropout is added to every layer in a CNN, the CNN sometimes gives a low accuracy on the training set. Hence it is imperative to balance the number of dropout layers with the training accuracy. However, it is to be noted that as mentioned in [1], if a dropout is not added after every layer, the CNN will not approximate a gaussian process. Hence we have to be very careful in adjusting the dropout layers and training accuracy.

4. **Extensive testing of this method is required:**This method is fairly new and this hasnt been tested extensively. For example I am not sure if this method would perform well, if the pool dataset is an out of distribution dataset. Most of my testing and the author's testing has been on in distribution dataset, which would not be the case in real world scenario.

7. Limitations:

In this section I will highlight a few limitations of my implementation.

1. The first limitation was that I had to make the CNN shallower to compensate for a weaker GPU.

2. The second limitation was that I could not comb through the entire pool set. This was because I did not have enough CPU power to go through 50,000 data points multiple times. hence I limited my datapoints to 3000 samples. Even after that it took more than 6 hours to train my model on CIFAR10 dataset.

8. Future Work:

As mentioned in the previous section, extensive testing of this method needs to be done to get a clear idea of the usability of this method in real world scenarios. Another possible avenue to explore would be to test this method using ensembles for uncertainity estimation[4]. The use of ensembles would limit the disadvantage mentioned in Section 6(point 1).

References

[1] 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. PMLR, 2016. 1, 2, 4, 5

[2] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR, 2017. 1, 2, 3, 4

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 2

[4] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017. 4, 5

[5] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 2

[6] X. Ren, H. Guo, S. Li, S. Wang, and J. Li. A novel image classification method with cnn-xgboost model. In *International Workshop on Digital Watermarking*, pages 378–390. Springer, 2017. 2

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 2