JOHANNES KEPLER
UNIVERSITY LINZ

Pascal Pilz

@ pasc.pilz@gmail.com

October 1, 2025

[Preprint] Deep Active Learning for Image Data

Practical Work in AI

Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Abstract

This report aims to reproduce the findings presented in Section 5.1 of Gal et al. [5], i.e., to show that Active Learning (AL) methods applied to Deep Leaning (DL) methods can outperform passive learning (random data acquisition). Gal et al. were among the first to combine AL and DL, marking a milestone in the history of AL.

We compare four acquisition functions to a random baseline and full-dataset training, using a Bayesian DL framework to apply AL to the MNIST dataset. We were largely able to reproduce the findings of Section 5.1 in Gal et al. [5], showing that AL can offer significant advantages over random data acquisition. However, we also observe differences between the results reported by Gal et al. and our findings and identify that the reporting of results in Gal et al. [5] is ambiguous in how those results were presented.

Contents

Abstract **2**

1. Introduction **4**
1.1 Active Learning 4
1.2 About this Report 4

2. Methods **5**
2.1 Bayesian CNN 5
2.1.1 Theory 5
2.1.2 Architecture 6
2.2 Acquisition functions and their approximations 6
2.2.1 Max entropy (ME) 7
2.2.2 Mutual Information (BALD) 7
2.2.3 Variation ratios (VR) 8
2.2.4 Mean standard deviation (MSTD) 8
2.2.5 Random 9
2.3 Dataset 9
2.4 Implementation Details 9

3. Experiments and results **10**
3.1 Experimental setup 10
3.2 Results 12
3.2.1 Test set accuracy and mutual information per acquisition step 12
3.2.2 Number of samples needed to reach certain test error 12

4. Limitations and future work **14**

5. Conclusion **17**

References **18**

Appendix A. Exact Results **20**

1. Introduction

1.1 Active Learning

Active Learning (AL) is a setting of Machine Learning (ML) where a *model* aims to achieve the best possible performance under a limited budget [21]. Instead of passively receiving data, the model actively selects additional samples to be labeled by an external *oracle*, for example a human annotator.

ML, and in particular Deep Learning (DL), has been used to great effect in recent times. A big challenge for DL applications is the availability of training data, since DL applications are often be extremely data hungry and the process of collecting and annotating data can be laborious and expensive. As such, the intersection between AL and DL —Deep Active Learning (DAL)— presents itself as a useful tool to tackle issues such as expensive data annotation or dataset distillation. DAL has been successfully used in fields such as agricultural development, medical diagnosis, microbiology, and manufacturing, for tasks such as named entity recognition, semantic parsing, object detection, image segmentation, text classification, and image analysis [5, 15, 23].

There are three fundamental forms of AL: query-, stream-, and pool-based [21]. In this work, we focus on pool-based AL. Formally, we are given an initial labeled training set $\mathcal{D}_l = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M}$ and a large pool of unlabeled data $\mathcal{D}_u = \{\mathbf{x}_j\}_{j=1}^{N}$, where $M \ll N$ and $y_i \in \{1, \dots, C\}$ is the class of $\mathbf{x}_i$. We then iteratively use a model $\mathcal{M}$ trained on the current training set together with an *acquisition function* (AF) $\alpha(\mathbf{x}, \mathcal{M})$ to choose a set of points $\mathcal{D}_q$ of batch size b from $\mathcal{D}_u$ according to $\mathcal{D}_q^* = \arg\max_{\mathbf{x} \in \mathcal{D}_u}^{b} \alpha(\mathbf{x}, \mathcal{M})$ (superscript b indicates taking the top b points), label them and added to the training set [23].

We call the process of acquiring new training samples from the pool an *acquisition step*, and we typically perform acquisition steps either for a certain number of steps, until a given budget is exhausted, or a desired performance is achieved. More detail on the exact procedure of an acquisition step can be found in section 3.1.

1.2 About this Report

In this report, we aim to reproduce the findings presented in Section 5.1 of Gal et al. [5]. Gal et al. were among the first to combine AL and DL for high-dimensional data, proposing to use Bayesian Convolutional Neural Networks [3] (BCNNs) on image data and showing that significant improvements over previous AL approaches be achieved. AL approaches prior to Gal et al. relied on techniques such as Support Vector Machines and Gaussian Processes, and struggled with issues related to scalability and high-dimensionality [5].

To do so, we compare four AFs against a random baseline and full-dataset training. Following Gal et al. [5], we perform three randomly initialized runs, each with 100 acquisition steps of 10 samples on the MNIST dataset [13]. For evaluation,

we track test set accuracy and the mutual information between target labels and model parameters at each acquisition step. In section 2.2 we describe the AFs, and in section 3.1 we outline the experimental setup and evaluation criteria.

The code for all experiments, as well the experiment parameters and results can be found at https://github.com/pilzpascal/2024s-bsc-project-active-learning.

2. Methods

In this section, we describe the specific techniques used. We will discuss the choice of model in section 2.1, acquisition functions in section 2.2, dataset in section 2.3, and concrete implementation details in section 2.4; which is modelled after Gal et al. [5], as well as based on input from our supervisor.

2.1 Bayesian CNN

Bayesian neural networks (BNNs) are robust to over-fitting, offer uncertainty estimates, and can learn from small amounts of data. This makes them well suited to the task of AL. Classical DL treats the model obtained after training as a point estimate, whereas Bayesian DL (BDL) provides a distribution over the parameters of the network, thus allowing us to extract information about model uncertainty from the posterior weight distribution [2, 3, 5, 10]. In this work, we use Bayesian Convolutional Neural Networks (BCNNs) [3].

2.1.1 Theory

Given a neural network $\mathbf{f}^{\omega}(\cdot)$ parameterized by a set of weights $\omega = \{\mathbf{W}_1, \dots, \mathbf{W}_L\}$, a new sample $\mathbf{x}^*$, and a training set $\mathcal{D}_{\text{tr}}$, the *posterior predictive distribution* is given by

$$p(y^*|\mathbf{x}^*, \mathcal{D}_{\text{tr}}) = \int_{\mathcal{W}} p(y^*|\mathbf{x}^*, \omega) p(\omega|\mathcal{D}_{\text{tr}}) \, d\omega,$$

where $\mathcal{W}$ is the space of all possible weight configurations for the chosen network architecture, and $p(\omega|\mathcal{D}_{\text{tr}})$ is the true posterior probability of a particular weight configuration $\omega \in \mathcal{W}$, which is typically intractable for neural networks [2, 3]. Since we deal with a classification task over C classes, we have

$$p(y|\mathbf{x}, \omega) = \text{softmax}\left(\mathbf{f}^{\omega}(\mathbf{x})\right).$$

Because of the intractability of the posterior weight distribution, we use a family of tractable probability distributions $q_\theta(\omega)$ to approximate the posterior, with $q_\theta^*(\omega)$ being the minimizer of the Kullback-Leibler (KL) divergence:

$$q_\theta^*(\omega) := \underset{q_\theta(\omega)}{\arg\min} \, \text{KL}\left[q_\theta(\omega) \| p(\omega|\mathcal{D}_{\text{tr}})\right].$$

The predictive distribution can then be approximated using *Monte-Carlo Dropout* [3, 4]:

$$\begin{aligned} p(y = c|\mathbf{x}, \mathcal{D}_{tr}) &= \int_{\mathcal{W}} p(y = c|\mathbf{x}, \omega) p(\omega|\mathcal{D}_{tr}) \, d\omega \\ &\approx \int p(y = c|\mathbf{x}, \omega) q_\theta^*(\omega) \, d\omega \\ &\approx \frac{1}{T} \sum_{t=1}^{T} p(y = c|\mathbf{x}, \hat{\omega}_t), \end{aligned}$$

with $\hat{\omega}_t \sim q_\theta^*(\omega)$, with $q_\theta(\omega)$ being the dropout distribution [5]. The first approximation stems from the fact that q_θ^* is chosen to be a minimizer of the KL divergence to the true posterior weight distribution, and the second approximation indicates a Monte-Carlo approximation of the integral.

In practice, this means that we train a neural network with dropout, resulting in a set of weights $\hat{\omega}$, which is a point estimate of the posterior weight distribution. During inference, we use dropout and perform T *stochastic forward passes* with different dropout masks, effectively generating T different sets of weights $\{\hat{\omega}_t\}_{t=1}^T$ used to obtain T different outputs $\{\text{softmax}(f^{\hat{\omega}_t}(\mathbf{x}))\}_{t=1}^T$ based on the same input $\mathbf{x}$.

In this work, we refer to each stochastic forward passes as an MC sample. For illustrative purposes, one can think of the MC samples as a $T \times C$ matrix, where the rows hold the softmax-outputs of the individual stochastic forward passes:

$$\begin{bmatrix} \text{softmax}(f^{\hat{\omega}_1}(\mathbf{x})) \\ \vdots \\ \text{softmax}(f^{\hat{\omega}_T}(\mathbf{x})) \end{bmatrix} = \begin{bmatrix} \hat{p}_1^1 & \hat{p}_2^1 & \cdots & \hat{p}_{C-1}^1 & \hat{p}_C^1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \hat{p}_1^T & \hat{p}_2^T & \cdots & \hat{p}_{C-1}^T & \hat{p}_C^T \end{bmatrix}$$

For sake of notation, we define $\hat{p}_c^t := p(y = c|\mathbf{x}, \hat{\omega}_t)$.

Intuitively, there is one such matrix per sample $\mathbf{x}$, and an acquisition function typically aims to reduce this matrix into a single number, indicating the *acquisition-value* of the given sample.

2.1.2 Architecture

The model architecture we use is based on LeNet-5 [12]; the exact specification can be found in Table 1.

2.2 Acquisition functions and their approximations

In this work, we compare the same four uncertainty-based acquisition functions as Gal et al. [5] and relate them to a random baseline, as well as to full-dataset training. Further details on their respective backgrounds and the derivations of their approximation can be found in [2, 5].

Table 1: LeNet-5 based model.

Order	Layer Type	Parameters
1	convolution + relu	6x5x5
2	max pooling	2x2
3	convolution + relu	16x5x5
4	max pooling	2x2
5	dropout	drop prob = 0.5
6	dense + relu	400x120
7	dropout	drop prob = 0.25
8	dense + relu	120x84
9	dense + relu	84x10

2.2.1 Max entropy (ME)

Max entropy is an information theory based method that chooses pool points which maximize the predictive entropy [22], meaning that this function is maximized when all classes have equal probability and minimized when one class has probability 1 and the other classes have probability 0 [2].

$$\mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{tr}}] = -\sum_{c} p(y = c|\mathbf{x}, \mathcal{D}_{\text{tr}}) \log p(y = c|\mathbf{x}, \mathcal{D}_{\text{tr}})$$

Given our setting of using Monte Carlo dropout, we can approximate the function using

$$\alpha_{\text{ME}}(\mathbf{x}, \mathcal{M}) := -\sum_{c} \left(\frac{1}{T} \sum_{t} \hat{p}_c^t \right) \log \left(\frac{1}{T} \sum_{t} \hat{p}_c^t \right). \tag{1}$$

2.2.2 Mutual Information (BALD)

Bayesian Active Learning by Disagreement (BALD) [7] uses the mutual information between the model predictions and the posterior of model parameters [2, 5, 23].

$$\mathbb{I}[y, \omega|\mathbf{x}, \mathcal{D}_{\text{tr}}] = \mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{tr}}] - \mathbb{E}_{p(\omega|\mathcal{D}_{\text{tr}})} [\mathbb{H}[y|\mathbf{x}, \omega]]$$

This function is maximized by a point $\mathbf{x}$ if the model on average is uncertain about the prediction y (high predictive entropy $\mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{tr}}]$), but for which model parameters exists that are erroneously more certain (low $\mathbb{E}_{p(\omega|\mathcal{D}_{\text{tr}})} [\mathbb{H}[y|\mathbf{x}, \omega]]$) [2].

Given our setting, we can approximate BALD using

$$\alpha_{\text{BALD}}(\mathbf{x}, \mathcal{M}) := -\sum_{c} \left(\frac{1}{T} \sum_{t} \hat{p}_c^t \right) \log \left(\frac{1}{T} \sum_{t} \hat{p}_c^t \right) + \frac{1}{T} \sum_{c,t} \hat{p}_c^t \log \hat{p}_c^t. \tag{2}$$

In the context of the above-mentioned matrix-model of MC dropout, one can think of BALD as looking for points that have high entropy in the class dimension, but low entropy in the MC samples dimension. I.e., all classes are closer to

equally likely when averaged across the MC samples dimension, while the individual stochastic forward passes produce rather distinct results, indicating that slight variations in model weights results in significant variation in model output.

Contrast this to ME, which only considers the entropy over the average of the MC samples, but not the entropy within each MC sample, i.e., only along the class dimension.

2.2.3 Variation ratios (VR)

Freeman [1] defines the variation ratio v as such: "v is an index of the amount of variation or the range of difference among the cases in a distribution of nominal scale classifications. [...] The mode is the best guess, and v is simply the proportion of wrong guesses." Accordingly, his formal definition is

$$v = 1 - \frac{f_{\text{mode}}}{N},$$

where f_{mode} is the frequency of the modal class, and N is the total number samples.

In our setting, we calculate it as

$$\alpha_{\text{VR}}(\mathbf{x}, \mathcal{M}) := 1 - \frac{1}{T} \max_c \sum_t \mathbb{1}[y_t = c], \tag{3}$$

where $\mathbb{1}[\cdot]$ is the indicator function and y_t is the class predicted for the forward pass using weights $\hat{\omega}_t$, i.e., $y_t = \arg\max_c f^{\hat{\omega}_t}(\mathbf{x})$. This effectively means that we get the relative frequency of the most often predicted class among the T MC samples, the modal frequency $\frac{1}{T} \max_c \sum_t \mathbb{1}[y_t = c]$, and calculate one minus this value. Relating it to Freeman [1], we can see that $\max_c \sum_t \mathbb{1}[y_t = c] = f_{\text{mode}}$ and $T = N$.

Gal et al. [5] define it as

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

noting in [2] that, under our assumptions on q_θ^* and given $T \to \infty$, we can derive how the formulation of Equation 3 can be interpreted as approximating $1 - \max_y p(y|\mathbf{x}, \mathcal{D}_{\text{tr}})$.

2.2.4 Mean standard deviation (MSTD)

Gal et al. [5] describe this method as more of an improvised approach. It was used by Kampffmeyer et al. [8] and Kendall et al. [9] to compute per-pixels uncertainty maps for image segmentation. They both do not give a theoretical justification, but Kendall et al. [9] explain that they chose MSTD over VR, the only other method they tried, since VR "qualitatively produce[s] a more binary measure of

model uncertainty."

$$\begin{aligned}\sigma(\mathbf{x}) &= \frac{1}{C}\sum_{c}\sqrt{\mathrm{Var}_{q(\omega)}[p(y=c|\mathbf{x},\omega)]} \\ &= \frac{1}{C}\sum_{c}\sqrt{\mathbb{E}_{q(\omega)}[p(y=c|\mathbf{x},\omega)^2] - \mathbb{E}_{q(\omega)}[p(y=c|\mathbf{x},\omega)]^2}\end{aligned}$$

In our setting, we can approximate this as

$$\alpha_{\mathrm{MSTD}}(\mathbf{x},\mathcal{M}) := \frac{1}{C}\sum_{c}\sqrt{\left(\frac{1}{T}\sum_{t}(\hat{p}_c^t)^2\right) - \left(\frac{1}{T}\sum_{t}\hat{p}_c^t\right)^2}. \tag{4}$$

Essentially, we simply calculate the standard deviation per class among the T MC samples, and then take the average over all C classes.

2.2.5 Random

$$\alpha_{\mathrm{random}}(\mathbf{x},\mathcal{M}) = \mathrm{unif}(), \tag{5}$$

where unif is a function that returns a uniformly distributed random number in $[0,1]$. Using this acquisition function is equivalent to choosing a pool point at random.

2.3 Dataset

Similar to Gal et al. [5], we use the MNIST dataset [13] for our experiments. The dataset consists of handwritten digits from 0 to 9 and contains $60,000$ training images and $10,000$ test images. Each image is 28x28 with one color channel and is accompanied by a label. An example of 30 samples taken from the training set can be seen in Figure 1.

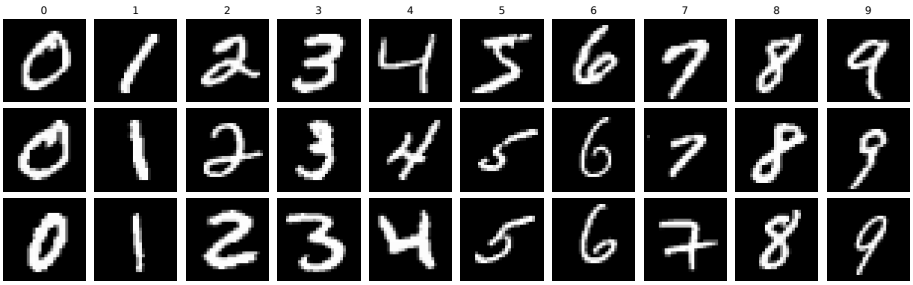

Figure 1: Examples from the MNIST training set, with their respective labels shown above.

2.4 Implementation Details

The code for all experiments, as well the experiment parameters and results can be found at https://github.com/pilzpascal/2024s-bsc-project-active-learning.

Implementation of the model, the training and AL loop, as well as the acquisition functions is done in Python (Version 3.12.4) using PyTorch [19] (Version 2.4.0). The experiments were conducted on a 2020 M1 MacBook Pro (macOS 15). The total runtime for all 18 experiments (3 repetitions x (5 acquisition functions + full-dataset training) = 18 experiments) was approximately 30 hours.

Experiments are being recorded using a YAML file. The experiment parameters are stored at the beginning of the file, the results get written to the file after each iteration of the AL loop.

3. Experiments and results

The goal of our experiments is to reproduce part of the results of Gal et al. [5]. In particular, to compare the four acquisition functions described in section 2.2 to each other, as well as to a random baseline. This corresponds to Section 5.1 in Gal et al. [5].

As a second baseline, in addition to Random, we train a model on the full training set of $60,000$ samples. Of these, $1,000$ samples are reserved for validation during early stopping, using the same validation set as in the AL loop for the corresponding experiment repetition.

This allows us to assess the efficiency and effectiveness of uncertainty-based sampling compared to random selection and full supervision.

3.1 Experimental setup

We use a Bayesian Convolutional Neural Network to run an AL loop on the MNIST dataset. The process starts with a small labeled dataset, the *initial training set*, to which iteratively new points will be added. The new points are selected from a large unlabeled dataset, the *pool set*. After each acquisition step, the model is retrained from scratch on the updated labeled set.

In total, we perform 100 *acquisition steps*, acquiring 10 new samples in each step to end up with a final training set of $1,020$ examples. Model performance is measured after every acquisition step using accuracy and mutual information.

What follows is a detailed explanation of experiment procedures:

Acquisition step Given a pool set, a training set, a randomly initialized model, and an acquisition function, each acquisition step consists of:

1. Train the untrained model on the current training set.
2. Measure the model performance on the test set using accuracy and mutual information.
3. Apply the trained model and the acquisition function to the pool points to assign an acquisition value to each point.

4. Select the 10 points with the highest acquisition values, remove them from the pool, and add them to the training set.

Due to computational limits, acquisition values were computed on a random subset of 5,000 pool samples at each step. Test accuracy was estimated on 3,000 randomly selected test samples for the same reason.

Data splits We split standard MNIST training set into three datasets:

- A random *validation set* of $1,000$ samples.
- A random but balanced *initial training set* of 20 samples, two of each class.
- A *pool set* consisting of the remaining $58,980$ samples.

As *test set*, we use the standard MNIST test set of $10,000$ samples.

Performance measure To measure model performance, we calculate the accuracy as well as mutual information over the test set after every acquisition step. Mutual information is calculated by applying Equation 2 to the entire pool and averaging the result.

Experiment repetitions We repeat each experiment three times. For reproducibility, a predefined sequence of three random seeds $\{1, 2, 3\}$ is used, ensuring that each acquisition function starts with the same initial training set in each repetition.

Number of MC samples We use $\mathsf{T} = 64$. Gal and Ghahramani [3] suggest $\mathsf{T} \geq 20$, but it is important to note that they found this number by evaluating the test error on the CIFAR-10 [11] using a Deeply Supervised Net [14]. Gal et al. [5] used $\mathsf{T} = 20$.

Training details We use the following training procedures and hyperparameters in each acquisition step:

Early stopping We train each model for 400 epochs and choose the model with the lowest validation loss. In this way, we do not use early stopping to save time, but only as a regularization technique.

Validation set We use a validation set of $1,000$-sample for early stopping. Initially, we tried a 100-sample validation set, as in Gal et al. [5], but this caused unreliable early stopping because of high variance and bias with small validation and training sets. This often led to stopping after the first epoch. To mitigate this, we increased the validation set to $1,000$ samples.

Loss function We use the categorical cross entropy loss.

Optimizer We choose the AdamW optimizer over Adam, as it correctly implements weight decay [16]. Other optimizers such as SGD and Adam were tested but showed no significant differences. AdamW hyperparameters: `learning rate = 0.001`, `betas = (0.9, 0.999)`, `weight decay = 0.01`.

Weight decay We use a weight decay of 0.01. Besides serving as a regularization technique, using weight decay is important as it corresponds to a Gaussian prior centered around 0 for the weights [3].

Batch size We use full batch training during acquisition steps, and mini-batches of 1,024 samples when training on the full dataset.

3.2 Results

In this section, we present our results and compare them to the results reported by Gal et al. [5]. For this, we simplify the results and do not report exact numbers in this section. For more precise results, refer to Appendix A.

3.2.1 Test set accuracy and mutual information per acquisition step

To evaluate test set accuracy and mutual information per acquisition step, we average the results of the three runs. Figure 2 shows the average test set accuracy and Figure 3 shows the average test set mutual information over the number of acquired samples, with the standard deviation across the three runs represented by the shaded area.

For test set accuracy, it appears that ME, BALD, VR, and MSTD perform similarly, with all of them meaningfully outperforming Random. This somewhat contradicts the results reported by Gal et al. [5], as they found MSTD to only slightly outperform Random. Our Figure 2 can be compared to Figure 1 by Gal et al. [5].

For test set mutual information, we can see that until around 400 acquired samples, ME performs worse than BALD, and VR, and MSTD, with the three of them performing about the same. After that, there appears to be no meaningful difference between the four acquisition functions. Random performs similar to ME, BALD, and VR until around 250 acquired samples, after that it performs better than the four acquisition functions.

We hypothesize that the better performance of Random can be explained by the fact that Random gathers a training set that is closer in distribution to the test set than with the other acquisition functions. Thus, the information to be gained from the test set is lower.

3.2.2 Number of samples needed to reach certain test error

In this section, we report on the number of samples required to reach 10% and 5% error rate, where error rate is defined to be `1-accuracy`. This serves as a direct numerical comparison of our results to those reported by Gal et al. [5]. To further facilitate a comparison, we normalize the results by the corresponding random baseline. For more precise values, refer to Appendix A.

Given our setting of three runs, one issue we face is that there are two different ways to obtain these values:

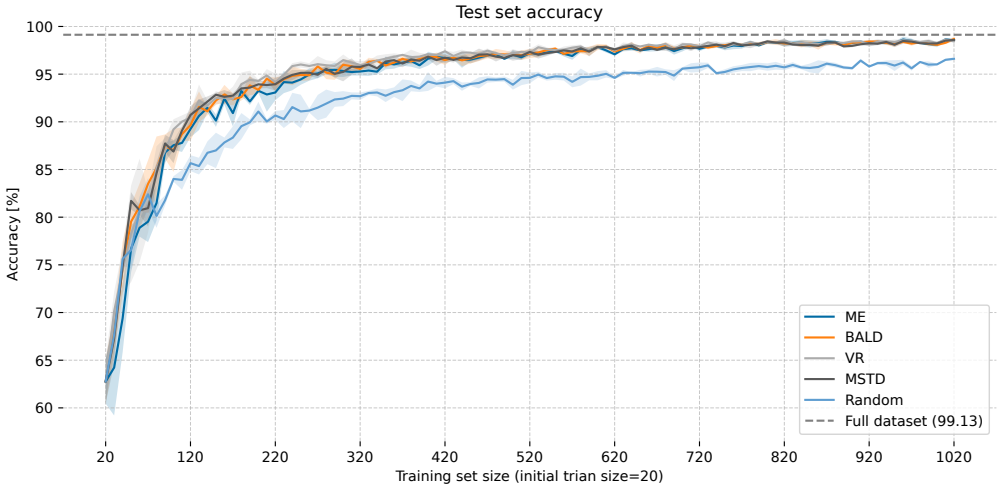

Figure 2: Mean across runs of test set accuracy over number of acquired samples. Standard deviation across runs is represented by the shaded area.

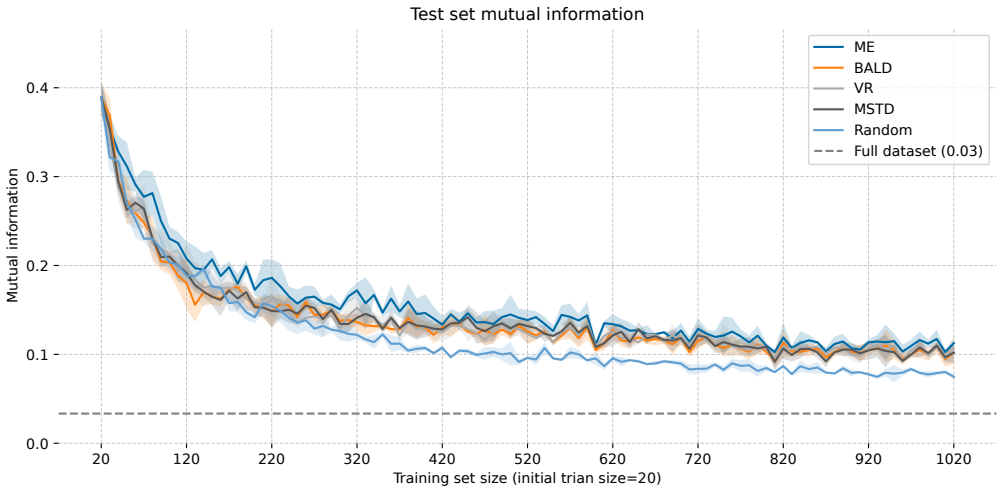

Figure 3: Mean across runs of test set mutual information over number of acquired samples. Standard deviation across runs is represented by the shaded area. Lower is better.

Step-Average: First determine the number of steps needed for each run individually and then average over the number of steps.

Average-Step: First average the accuracy over the three runs and then determine the number of steps needed for this average run.

(We refer to them as `Step-Average` and `Average-Step`, as this aligns with their respective order of operation: first determine the number of *steps* and then *average* this number, or first *average* the runs and then determine the number of *steps*.)

`Step-Average` allows us to calculate the standard deviation as a measure of variability across runs, while `Average-Step` does not allow for this. `Average-Step` lines up with the results we report in section 3.2.1, as there we directly average the accuracy of the three runs. The fact that the numbers reported by Gal et al. [5] in their Table 1 are integers indicates to us that they used `Average-Step`, as this always results in an integer value, whereas `Step-Average` is likely to result in a decimal value. Therefore, to have a direct comparison to Gal et al. [5], we show the results of `Average-Step` and for completeness we show the results of `Step-Average`.

It is important to note that, based on visual inspection, `Step-Average` appears to generally result in fewer samples required, especially for acquisition functions that have a higher variance between runs. This can be seen in Figure 4, which shows the three runs of Random as well as the average run. Each of the three runs reaches the threshold earlier, while the average run achieves it latest.

Figure 5 shows a comparison of our results and those of Gal et al. [5] regarding the number of samples needed to reach a test error of 10% and 5%. For easier comparison, we show a relative form of these results in Figure 6. To relativize the results, the number of samples needed is normalized through dividing it by the corresponding random baseline. For example, in our experiments, reaching a 10% error rate for BALD required 130 samples and for Random 200. After normalization, this means that BALD achieved 10% error rate using 0.65 as many samples as Random.

In Figure 6 we can observe that our results appear similar to those of Gal et al. [5] (their results are taken from their Table 1). An obvious difference we found is that MSTD performs rather similar to the other acquisition function, whereas Gal et al. [5] reports it to be underperforming. In Figure 5 we can also see that using `Step-Average` to calculate the number of steps needed to reach 5% error rate for Random has a high variance, resulting in the number returned by `Step-Average` and `Average-Step` being significantly different (also illustrated in Figure 4), meaning that normalizing by this value skews the data depicted in Figure 6 slightly.

4. Limitations and future work

Model. The LeNet-5 based model we used for the experiments has 61,706 parameters, whereas the model used by Gal et al. [5] is based on the example Keras

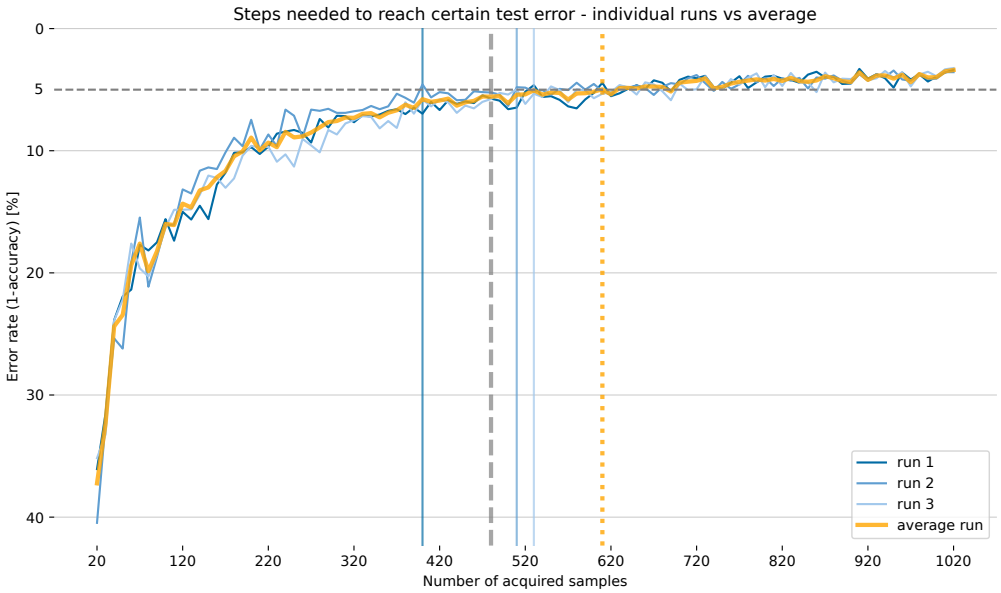

Figure 4: Random baseline test set error rate of the three individual runs and their average run. The solid vertical lines show the points where the respective runs cross the 5% threshold. The dashed line shows the average of the solid lines, representing `Step-Average`, while the dotted line shows where the average run crosses the threshold, representing `Average-Step`.

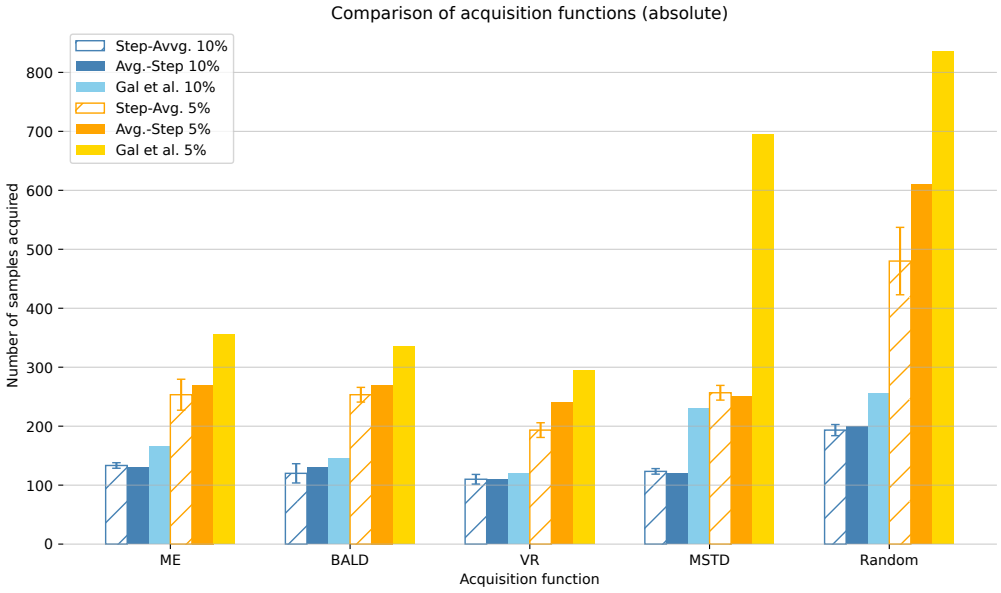

Figure 5: Number of samples required to achieve 10% and 5% error rate. The error bars indicate the standard deviation (only applicable for `Step-Average`). Lower is better.

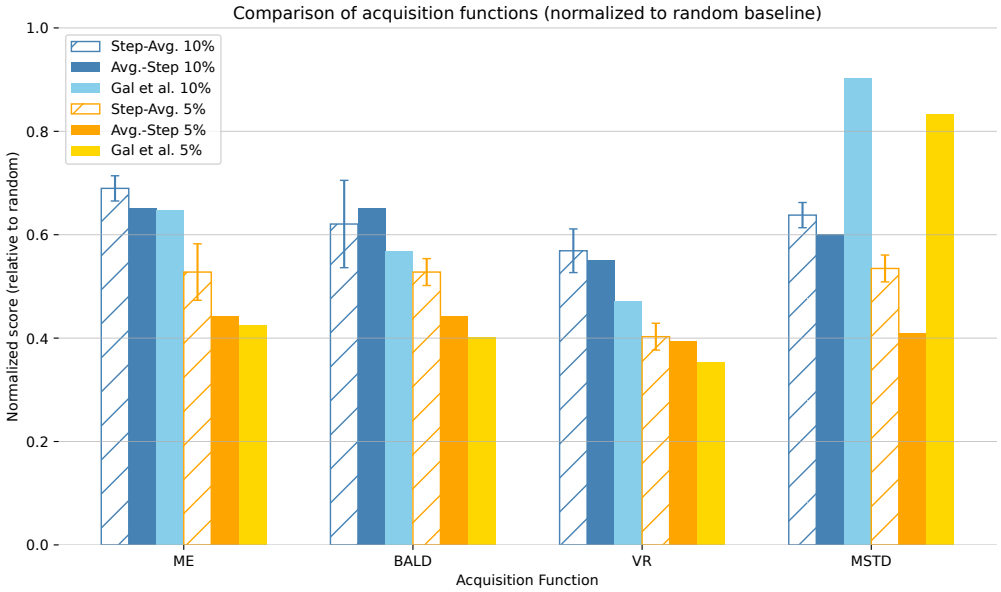

Figure 6: Number of samples required to achieve 10% and 5% error rate, normalized to the respective random baseline. The error bars indicate the standard deviation (only applicable for `Step-Average`). Lower is better.

MNIST CNN implementation and has $608,202$ parameters. Given the limited computational resources available to us, we could not replicate the experiments of Gal et al. [5] using the same model they used. Accordingly, our results might not be directly comparable. A more powerful model could better be able to learn from small amounts of data, but could also more easily over-fit on a small training set, which is a relevant concern in the AL setting.

Pool and test subsets. Due to our computational constraints, we opted to use a subset of the pool and the test set in each acquisition step. Further details on our procedure can be found in section 3.1. A similar approach was also used by [17].

Batch acquisition. Kirsch et al. [10] introduce BatchBALD, an approach to acquire a batch of data which considers dependencies within the batch. BatchBALD is an alternative to BALD that takes as input not just a single sample, but a set of samples and assigns a scalar acquisition-value to the entire set. They show that the naive approach of taking the top-k samples with the highest acquisition-value can yield similar and redundant samples, and that employing their new strategy can give drastic performance improvements.

Uncertainty estimation. In this work, we use MC Dropout [4] to estimate the posterior weight distribution, as this method was also used by Gal et al. [5]. MC Dropout is one of the most popular methods due to its simplicity and ease of use [6].

It would be interesting to explore other methods, for example Mobiny et al. [18] suggest MC-DropConnect, a method similar to MC Dropout that drops individual connections between units instead of whole units. Another, more recent method by Schweighofer et al. [20] searches for adversarial models to estimate predictive uncertainty, i.e., a model that explains the training data similarly well, but gives different predictions for test data points.

5. Conclusion

In this work, we set out to reproduce the experiments and findings of Section 5.1 in Gal et al. [5]. To do so, we recreated their experimental setup as closely as we were able to under the given computational constraints.

A methodological issue we found is that there are at least two ways of calculating how many samples were needed to reach a given test error: `Step-Average` and `Average-Step`. Depending on the chosen approach, the results can vary greatly, and Gal et al. [5] were not clear about which approach they employed.

A major difference we found regarding the performance of the acquisition functions concerns MSTD. Gal et al. [5] reports MSTD to perform similar to Random, greatly underperforming ME, BALD, and VR. In contrast to this, we found MSTD to perform closer to the other acquisition functions.

Overall, we were largely able to reproduce the findings of Gal et al. [5], showing that the studied uncertainty based AL methods can greatly increase data efficiency of high dimensional image data classification tasks.

References

[1] Linton C Freeman. 1965. *Elementary Applied Statistics: For Students in Behavioral Science*.

[2] Yarin Gal. 2016. Uncertainty in Deep Learning.

[3] Yarin Gal and Zoubin Ghahramani. 2016. Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. (January 18, 2016). arXiv: 1506.02158 [stat]. Retrieved 03/11/2025 from http://arxiv.org/abs/1506.02158. Pre-published.

[4] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, (October 4, 2016).

[5] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. 2017. Deep Bayesian Active Learning with Image Data. (March 8, 2017). arXiv: 1703.02910 [cs, stat]. Retrieved 03/07/2024 from http://arxiv.org/abs/1703.02910. Pre-published.

[6] Wenchong He, Zhe Jiang, Tingsong Xiao, Zelin Xu, and Yukun Li. 2025. A Survey on Uncertainty Quantification Methods for Deep Learning. (January 20, 2025). arXiv: 2302.13425 [cs]. Retrieved 09/16/2025 from http://arxiv.org/abs/2302.13425. Pre-published.

[7] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. 2011. Bayesian Active Learning for Classification and Preference Learning. (December 24, 2011). arXiv: 1112.5745 [cs, stat]. Retrieved 07/10/2024 from http://arxiv.org/abs/1112.5745. Pre-published.

[8] Michael Kampffmeyer, Arnt-Borre Salberg, and Robert Jenssen. 2016. Semantic Segmentation of Small Objects and Modeling of Uncertainty in Urban Remote Sensing Images Using Deep Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). IEEE, Las Vegas, NV, USA, (June 2016), pp. 680–688. ISBN: 978-1-5090-1437-8. DOI: 10.1109/CVPRW.2016.90. Retrieved 07/30/2025 from http://ieeexplore.ieee.org/document/7789580/.

[9] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. 2016. Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. (October 10, 2016). arXiv: 1511.02680 [cs]. Retrieved 07/30/2025 from http://arxiv.org/abs/1511.02680. Pre-published.

[10] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. 2019. BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning. (October 28, 2019). arXiv: 1906.08158 [cs]. Retrieved 08/23/2025 from http://arxiv.org/abs/1906.08158. Pre-published.

[11] Alex Krizhevsky. [n. d.] Learning Multiple Layers of Features from Tiny Images.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86, 11, 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791. Retrieved 07/24/2025 from http://ieeexplore.ieee.org/document/726791/.

[13] Yann LeCun and Corinna Cortes. The MNIST Database of Handwritten Digits.

[14] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2014. Deeply-Supervised Nets. (September 25, 2014). arXiv: 1409.5185 [stat]. Retrieved 08/12/2025 from http://arxiv.org/abs/1409.5185. Pre-published.

[15] Dongyuan Li, Zhen Wang, Yankai Chen, Renhe Jiang, Weiping Ding, and Manabu Okumura. 2025. A Survey on Deep Active Learning: Recent Advances and New Frontiers. *IEEE Transactions on Neural Networks and Learning Systems*, 36, 4, (April 2025), 5879–5899. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/tnnls.2024.3396463. Retrieved 07/21/2025 from https://ieeexplore.ieee.org/document/10537213/.

[16] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. (January 4, 2019). arXiv: 1711.05101 [cs]. Retrieved 08/22/2025 from http://arxiv.org/abs/1711.05101. Pre-published.

[17] lunayht. 2025. Lunayht/DBALwithImgData. (February 25, 2025). Retrieved 09/08/2025 from https://github.com/lunayht/DBALwithImgData.

[18] Aryan Mobiny, Hien V. Nguyen, Supratik Moulik, Naveen Garg, and Carol C. Wu. 2019. DropConnect Is Effective in Modeling Uncertainty of Bayesian Deep Networks. (June 7, 2019). arXiv: 1906.04569 [cs]. Retrieved 07/22/2025 from http://arxiv.org/abs/1906.04569. Pre-published.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. (December 3, 2019). arXiv: 1912.01703 [cs]. Retrieved 09/25/2025 from http://arxiv.org/abs/1912.01703. Pre-published.

[20] Kajetan Schweighofer, Lukas Aichberger, Mykyta Ielanskyi, and Günter Klambauer Sepp Hochreiter. 2023. Quantification of Uncertainty with Adversarial Models.

[21] Burr Settles. 2009. Active Learning Literature Survey, (January 26, 2009).

[22] C E Shannon. 1948. A Mathematical Theory of Communication.

[23] Xueying Zhan, Qingzhong Wang, Kuan-hao Huang, Haoyi Xiong, Dejing Dou, and Antoni B. Chan. 2022. A Comparative Survey of Deep Active Learning. (July 19, 2022). arXiv: 2203.13450 [cs]. Retrieved 03/07/2024 from http://arxiv.org/abs/2203.13450. Pre-published.

Appendix A. Exact Results

In section 3.2.2 we discussed that, given multiple runs, there are at least two different approaches to determine how many samples are needed to reach a certain accuracy:

`Step-Average`: First determine the number of steps needed for each run individually and then average over the number of steps.

`Average-Step`: First average the accuracy over the runs and then determine the number of steps needed for this average run.

(We refer to them as `Step-Average` and `Average-Step`, as this aligns with their respective order of operation: first determine the number of *steps* and then *average* this number, or first *average* the runs and then determine the number of *steps*.)

In this section, we report the exact number of steps needed to reach 10% and 5% test error. Table 2 and Table 3 show the absolute numbers for 10% and 5% respectively, while Table 4 and Table 5 show the relative numbers for 10% and 5% respectively, i.e., normalized to the random baseline, as was detailed in section 3.2.2. Each table shows the results of both `Step-Average` and `Average-Step`, as well as the results reported by Gal et al. [5] in their Table 1. Note that the individual runs were sorted, and that decimal numbers were rounded to two places (three for standard deviations).

The complete experiment results, i.e., the test set accuracy and mutual information for all three runs, all five acquisition functions for each acquisition step can be found in https://github.com/pilzpascal/2024s-bsc-project-active-learning..

Table 2: Number of samples required to reach 10% test error, for both our experiments and Gal et al. [5]. **Bold** indicates the best (lowest).

Acq. Func.	Step-Average		Average-Step	
	Runs	**Mean±Std**	**Ours**	**Gal et al.**
ME	[130, 130, 140]	133.3±4.7	130	165
BALD	[100, 120, 140]	120.0±16.3	130	145
VR	[100, 110, 120]	**110.0**±8.2	**110**	**120**
MSTD	[120, 120, 130]	123.3±4.7	120	230
Random	[180, 200, 200]	193.3±9.4	200	255

Table 3: Number of samples required to reach 5% test error, for both our experiments and Gal et al. [5]. **Bold** indicates the best (lowest).

Acq. Func.	Step-Average		Average-Step	
	Runs	**Mean±Std**	**Ours**	**Gal et al.**
ME	[230, 240, 290]	253.3±26.2	270	355
BALD	[240, 250, 270]	253.3±12.5	270	335
VR	[180, 190, 210]	**193.3**±12.5	**240**	**295**
MSTD	[240, 260, 270]	256.7±12.5	250	695
Random	[400, 510, 530]	480.0±57.2	610	835

Table 4: Relative number of samples required to reach 10% test error, for both our experiments and Gal et al. [5]. **Bold** indicates the best (lowest).

Acq. Func.	Step-Average		Average-Step	
	Runs	**Mean±Std**	**Ours**	**Gal et al.**
ME	[0.67, 0.67, 0.72]	0.69±0.024	0.65	0.65
BALD	[0.52, 0.62, 0.72]	0.62±0.084	0.65	0.57
VR	[0.52, 0.57, 0.62]	**0.57**±0.042	**0.55**	**0.47**
MSTD	[0.62, 0.62, 0.67]	0.64±0.024	0.60	0.90
Random	[0.93, 1.03, 1.03]	1.00±0.049	1.00	1.00

Table 5: Relative number of samples required to reach 5% test error, for both our experiments and Gal et al. [5]. **Bold** indicates the best (lowest).

Acq. Func.	`Step-Average`		`Average-Step`	
	Runs	**Mean±Std**	**Ours**	**Gal et al.**
ME	[0.48, 0.50, 0.60]	0.53±0.055	0.44	0.43
BALD	[0.50, 0.52, 0.72]	0.53±0.026	0.44	0.40
VR	[0.38, 0.40, 0.44]	**0.40**±0.026	**0.39**	**0.35**
MSTD	[0.50, 0.54, 0.56]	0.53±0.026	0.41	0.83
Random	[0.83, 1.06, 1.10]	1.00±0.119	1.00	1.00